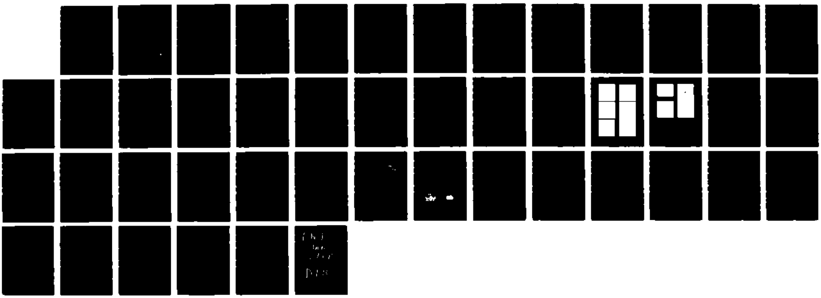
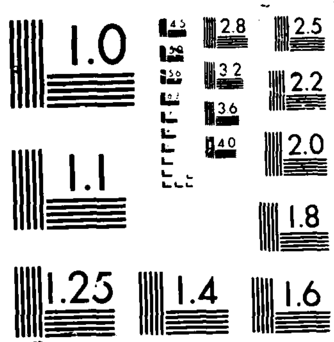
AD-A187 014 FRACTAL IMAGES: PROCEDURE AND THEORY(U) HARRY DIAMOND LABS ADELPHI MD S D CASEY AUG 87 HDL-TR-2119 1/1

UNCLASSIFIED F/G 12/5 NL




1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

HDL-TR-2119

August 1987

Fractal Images: Procedure and Theory

by Stephen D. Casey

U.S. Army Laboratory Command
Harry Diamond Laboratories
Adelphi, MD 20783-1197

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

2

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS AD-A187 014
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) HDL-TR-2119	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Harry Diamond Laboratories 6b. OFFICE SYMBOL (If applicable) SLCHD-RT-RB	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) 2800 Powder Mill Road Adelphi, MD 20783-1197	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION U.S. Army Laboratory Command 8b. OFFICE SYMBOL (If applicable) AMSLC	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 2800 Powder Mill Road Adelphi, MD 20773-1145	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

DTIC ELECTE DEC 1 0 1987 S E

11. TITLE (Include Security Classification)

Fractal Images: Procedure and Theory

12. PERSONAL AUTHOR(S)
Stephen D. Casey

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM May 86 TO Dec 86	August 1987	44

16. SUPPLEMENTARY NOTATION

HDL Project: 59B650

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Dynamical system, squig fractal, seed fractal
12	01		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The paper gives a study on the theory and production of fractals. It focuses on three areas in which fractals appear—dynamical systems, scaling, and random motion. In the section on dynamical systems, the dynamics of functions of one complex variable are discussed. The section on scaling includes a discussion of seed fractals, as well as a discussion of fractional dimension. The section on random motion is a discussion of squig fractals. In each section, sample images and computer code/pseudocode are provided. The paper concludes with application of fractals to digital signal processing.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Stephen D. Casey	22b. TELEPHONE (Include Area Code) (202) 394-2520 22c. OFFICE SYMBOL SLCHD-RT-RB

DD FORM 1473, 84 MAR   83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

87 11 27 246

CONTENTS

FIGURES

CODE LISTINGS

1. INTRODUCTION: A BIT OF HISTORY

"...no one doubts that the modern formulations (of science) are clear, elegant, and precise; it's just that it's impossible to comprehend how anyone ever thought of them."

--Michael Spivak, A Comprehensive Introduction to Differential Geometry

In the last few years, the word "fractal" (a set with noninteger fractional dimension) has worked its way out from research into more general use in the scientific community. The word was coined in the seventies by Benoit B. Mandelbrot to describe the elaborate images he was producing on computers at the Watson IBM Research Center. However, as Mandelbrot himself points out in his book, The Fractal Geometry of Nature (1983),* the roots of the fractal idea reach back over a century.

In the nineteenth century, mathematicians were working to develop rigor in their study of mathematics. They encountered curious phenomena, which forced the rethinking of some concepts. Four of these phenomena are cited:

1. In 1874, K. Weierstrass produced a continuous, but nowhere differentiable function. To create this function, he employed trigonometric series and lacunary (or "gap") series. An example of a Weierstrass function (fig. 1) is given by the formula

$$f(t) = \sum_{n} \left[\left(\frac{1}{2}\right)^{n} \right] \cos(2^{n} t) \quad .$$

(Weierstrass' original results were more general.)

2. In the 1870's, G. Cantor produced several results which gave relationships between set theory and calculus. In 1874, he produced his proof that there are only countably many algebraic numbers, and became increasingly fascinated with the concept of infinity in mathematics. A concrete example of his ideas is the "middle thirds" set, which was used by H. Lebesgue in the 1920's to produce the Cantor-Lebesgue function (fig. 2). This singular function has a derivative equal to zero at almost all points in the set (0,1), yet is monotonically increasing.

3. In 1890, G. Peano produced an example of a continuous space-filling curve (fig. 3). This curve maps the unit interval [0,1] onto the unit square $[0,1] \times [0,1]$.

4. In 1904, H. von Koch produced his snowflake (fig. 4). This curve has infinite length, but is contained in a finite area. The snowflake is nonintersecting and is self-similar, i.e., it appears the same despite successive magnifications.

In each of the above examples, a rigorous limit procedure was used (uniform convergence in the proper topology). However, because each of these examples conflicted with the geometric intuition of the time, they were

*See bibliography.

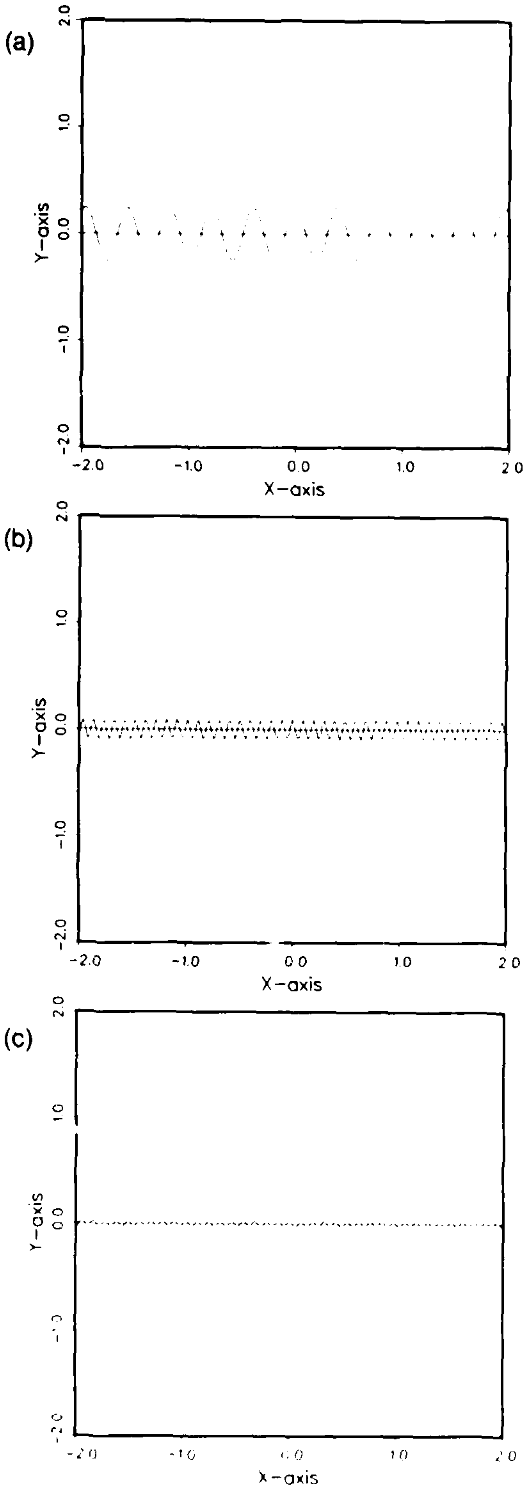


Figure 1. Weierstrass function: defining procedure followed (a) four times, (b) six times, and (c) eight times.

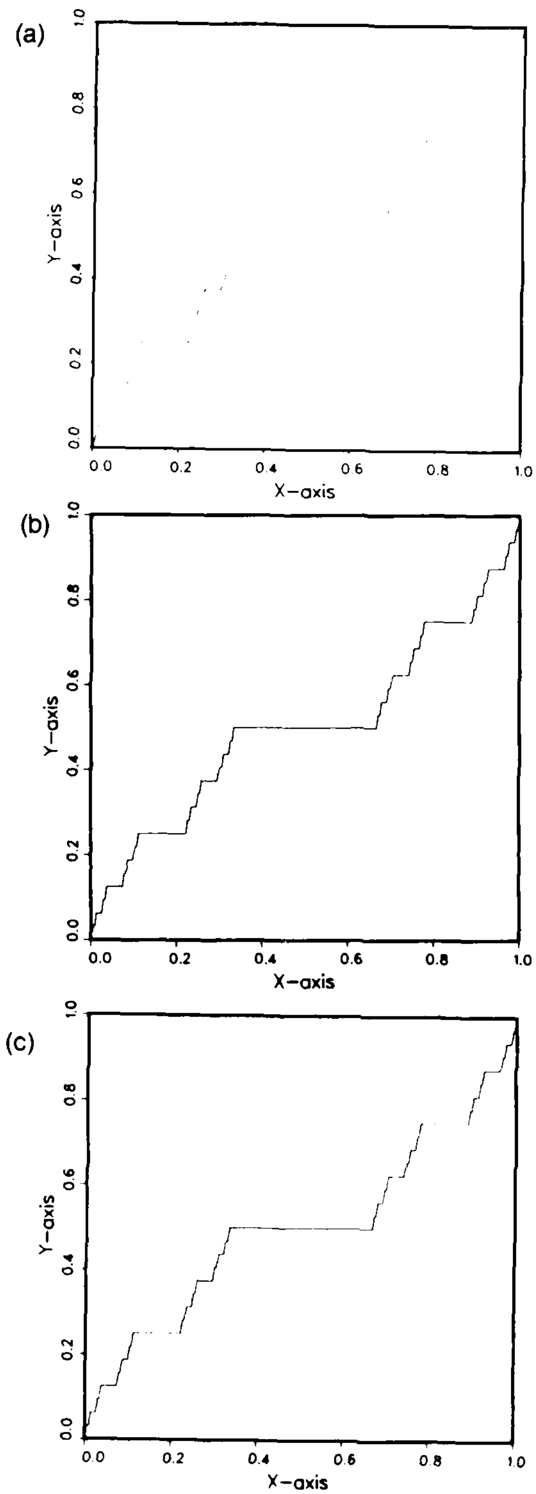


Figure 2. Cantor-Lebesgue function: defining procedure followed (a) three times, (b) five times, and (c) seven times.

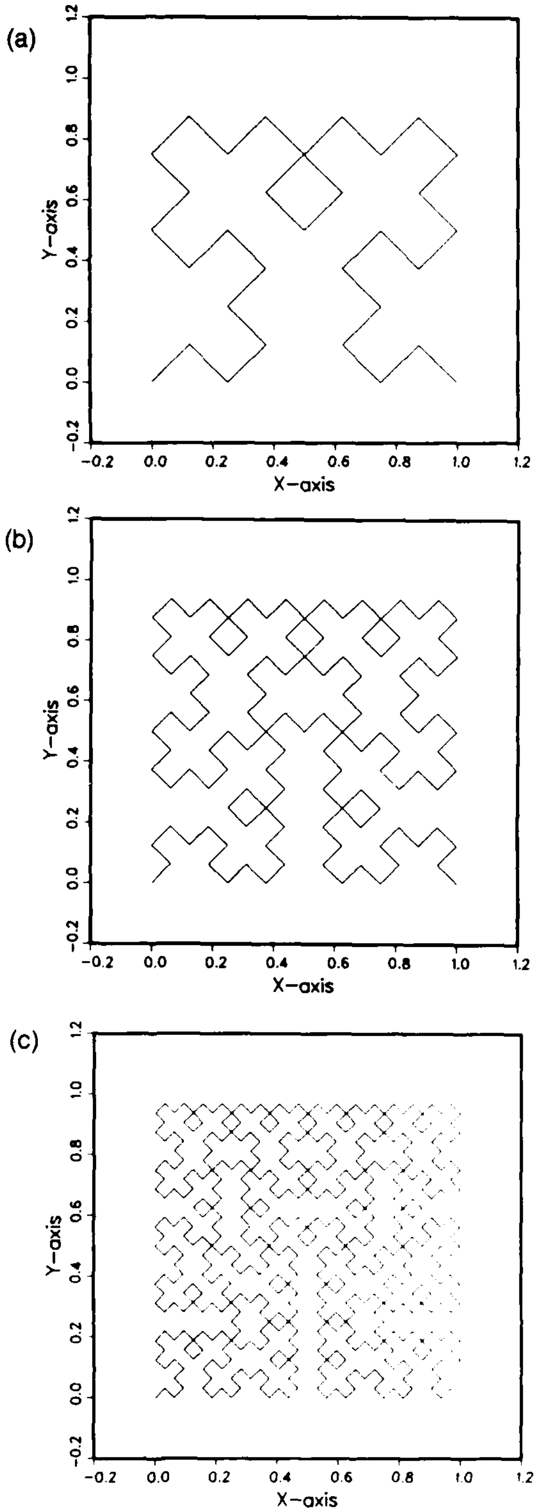


Figure 3. Peano curve: defining procedure followed (a) three times, (b) four times, and (c) five times.

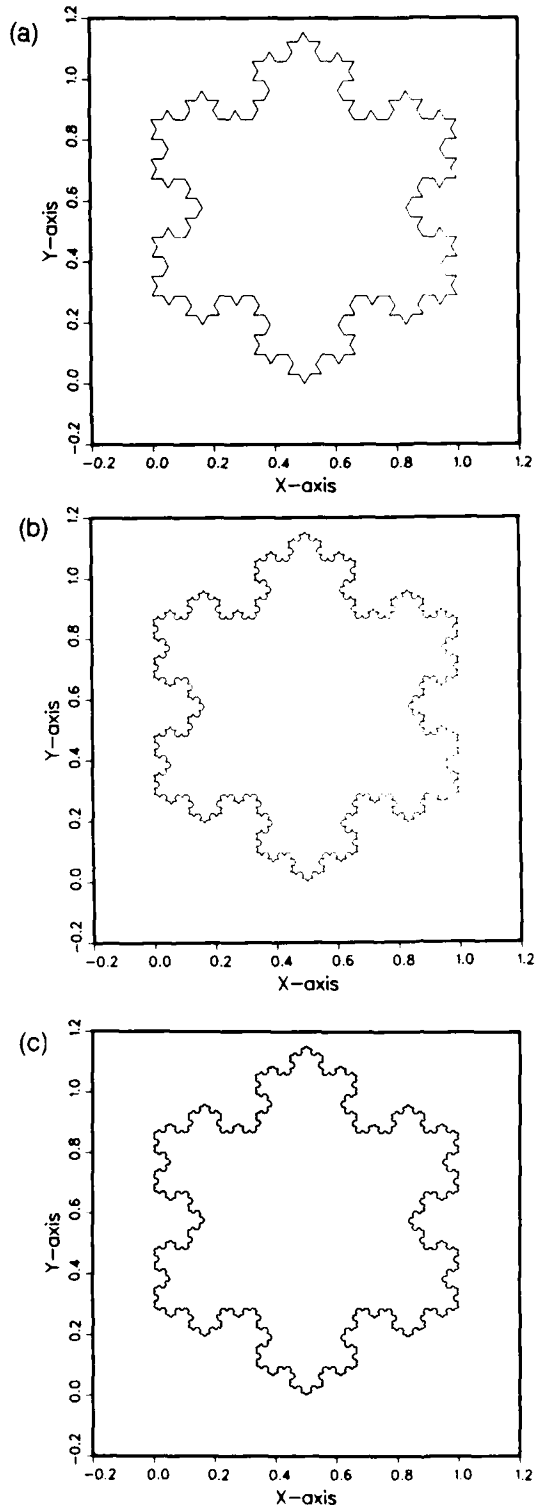


Figure 4. Koch's snowflake: defining procedure followed (a) three times, (b) four times, and (c) five times.

labeled "pathological." Yet, because of further scientific research, and especially because of the introduction of the computer, it is now possible to see that these examples appear to model nature quite well. Each curve fits Mandelbrot's basic definition of a fractal curve: a curve having fractional dimension higher than one. Mandelbrot cites all of them as important examples in his book.

In the following sections, theory and procedure for the creation of fractal images are discussed. The discussion is aimed at the reader with some computer graphics experience and/or software. Numerous fractal images are included, and algorithms for the production of some of these images are provided. The algorithms can be set up to run on almost any system--whenever possible, hardware specifics are eliminated. Thus, readers can run these algorithms on their own systems and proceed to explore the world of fractals. Discussion is divided into three areas: fractals on complex dynamical systems, seed fractals, and squig fractals.

The theoretical discussions in the following sections are independent of the descriptions of algorithms which produce fractal images. In particular, theoretical discussions precede algorithms in section 2 (on complex analytic dynamics). These were provided to give the interested reader some insight as to why the algorithms work. Also, section 3.2 (on dimension) must be labeled "theoretical," as discussion and calculation here are rather involved. Theoretical sections can be skipped over with a minimal loss of continuity.

2. COMPLEX ANALYTIC DYNAMICS

Many pioneers besides those already mentioned were involved in the evolution of the fractal--Riemann, Hausdorff, Klein, Cesaro, and Bernoulli, to name a few--the list is very long. One research field related to fractals that has had quite a revival lately is the field of iteration theory, or complex analytic dynamics. This was started in the early 1900's by P. Fatou and G. Julia. Both wrote long monographs on the subject. Today, its pioneers include D. Sullivan, J. Hubbard, and A. Douady.

2.1 Theory

A dynamical system consists of a pair, (X, Φ), where X is a topological space,* and $\Phi = \{\phi_t\colon t \in R\}$ is a set of dynamics, i.e., rules for the evolution of the system in time. If ϕ_t is continuous, the pair (X, ϕ_t) is usually called a flow.

Examples of dynamical systems appear in numerous places: the cardiovascular system, the capitalist system, and the solar system are all dynamic. In each, a process is occurring which can be thought of as evolving in time.

*A topological space is a set in which the concept "open set" is well defined. These topics are discussed by Munkres (1975), bibliography.

In this section, the underlying space of the dynamical system is the Riemann sphere, $\hat{C}$, where

$$\hat{C} = C \cup \{\infty\} = \{z = x + iy\colon x, y \in R,\ i = \sqrt{-1}\} \cup \{\infty\} \quad .$$

The point "∞" is added to C, the complex plane, by rolling the plane up into a sphere, and letting ∞ be the north pole. The dynamic is an analytic function.* Let f(z) denote this function. Then, by iterating the function

$$z_{n+1} = f(z_n) \quad ,$$

one gets a dynamical system. Note that this system is discrete.

The computer has proven to be a most useful tool in the study of nonlinear dynamical systems. This study has produced as a byproduct some of the fractal images seen here. Of these, many have been produced by complex analytic dynamics. Probably the most common dynamic in generating these images has been the now-famous equation

$$f(z) = z^2 + \lambda \quad .$$

Using this equation, we can discuss the two different types of images it produces. The first type is a $\hat{C}$-dynamical system. In this process, the number λ is kept constant, and z is varied. The second type of image is a parameter space image. Here, z is fixed, and the number λ is varied. Each different value of λ parameterizes a dynamical system from $\hat{C}$. The Mandelbrot set, which is generated in this fashion using $z^2 + \lambda$, is a parameter space image.

The following discussion of some of the theory behind generating these images comes under the category "complex quadratic dynamics." Blanchard (1984) gives a more complete discussion (see bibliography, "Mathematics," for other necessary background).

First, consider the dynamics in $\hat{C}$. Heuristically, the Riemann sphere, instead of just the complex plane, C, is the base space because of the importance of infinity in dynamics. In $\hat{C}$, the group of one-on-one analytic mappings is the group of Moebius transforms, that is, maps of the form

$$g(z) = \frac{az + b}{cz + d} \quad , \quad ad - bc \neq 0 \quad .$$

Moebius transformations have one zero and one pole. Using these maps, it is possible to get some insight into why the single equation $f(z) = z^2 + \lambda$ is so powerful. Let $h(z) = Az^2 + 2Bz + C$ be a general quadratic equation in $\hat{C}$. For f(z) as above, it is possible to solve for a Moebius transform g(z) and a value of λ in f(z) such that

$$h(z) = g^{-1} \circ f \circ g(z) = g^{-1}(f(g(z))) \quad .$$

*A function is called analytic in some open set if it can be expanded in a Taylor series in that set. In C-analysis, if a function has a continuous derivative in an open set, it is analytic there.

The solution is given by

$$g(z) = Az + B \quad ,$$

$$\lambda = AC + B - B^2 \quad .$$

Since two dynamical systems in $\hat{C}$ conjugate by a Moebius transform are the same, every quadratic dynamical system in $\hat{C}$ can be obtained by varying λ. In other words, every quadratic system is parameterized by the complex number λ.

Given the dynamic $f(z) = z^2 + \lambda$ for fixed λ, iteration produces a dynamical system. Under this iteration, individual points will have neighborhoods (small open sets containing the point in discussion) that exhibit one of two types of behavior. Points in these neighborhoods will either converge to a point after repeated iteration, or they will not converge. Those points that have a neighborhood of points that converge are called elements of the Fatou set. The points not in the Fatou set are called elements of the Julia set.

The function $f(z) = z^2$ provides an enlightening example of this dichotomy. Note that under iteration of f, every point with absolute value strictly less than one will converge to zero, while every point z with $|z| > 1$ will converge to infinity. However, under

$$f_n = f \circ \ldots \circ f = f(f(\ldots(f(z)))) \quad , \quad n \text{ times},$$

most points on the unit circle ($\{|z| = 1\}$) are just "spun around" at a faster and faster rate. In fact, if $z = \exp(i\alpha)$, where α is an irrational number, there exists an iterate of z coming quite close to any point on the unit circle. Thus, although the point 1 remains fixed under iterates of f, there always exists a point close by that will be moved somewhere else under iteration. A similar fate falls upon all points in $\{|z| = 1\}$, and thus it is possible to see that the Julia set of $f(z) = z^2$ is $\{|z| = 1\}$.

By adding a constant λ with relatively small absolute value, the dynamical system produced by $f(z) = z^2 + \lambda$ will still have a Julia set that is a simple closed curve. However, this curve exhibits a quasi-self-similarity, i.e., it is a fractal. As the value of $|\lambda|$ gets larger, the curve degenerates and no longer has a nicely defined inside and outside. (Fig. 5 shows various Julia sets produced by different functions.)

This behavior shows up in the parameter space image of $z^2 + \lambda$ and is represented by the Mandelbrot set. Recall that this set is generated by fixing z and varying λ. By definition, the Mandelbrot set is the set of complex numbers for which the dynamical system generated by $f(z) = z^2 + \lambda$ has a connected Julia set. For $\lambda = 0$, the Julia set is $\{|z| = 1\}$. As $|\lambda|$ increases, the resultant Julia set of the dynamical system generated by f will degenerate from a simple closed curve. When $|\lambda|$ gets sufficiently large, the attractive basis inside the curve bifurcates, i.e., splits. When this occurs, the boundary of the Mandlebrot set has been reached. Theoretically, it has been proven that this behavior is completely determined by the growth of $z = 0$

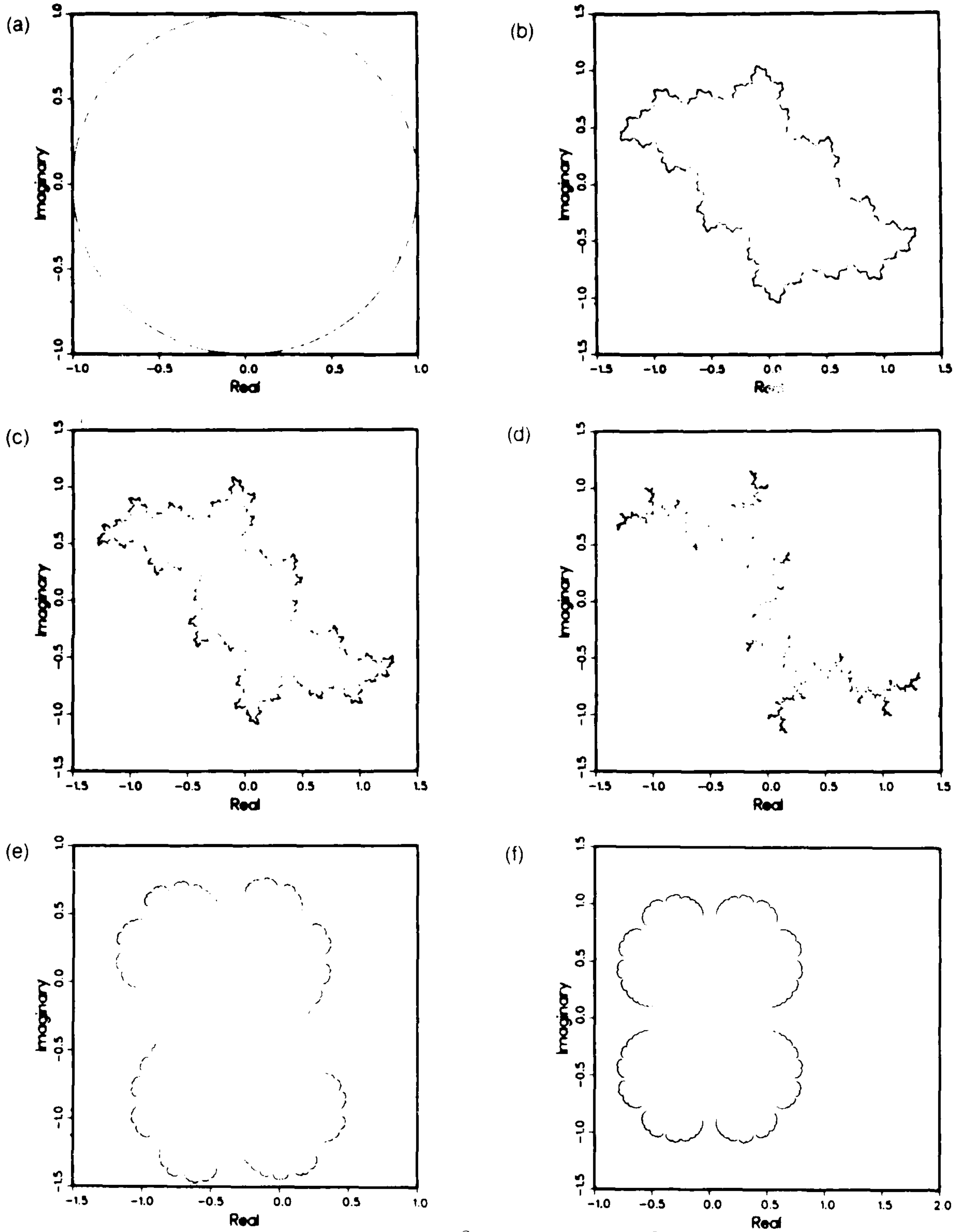


Figure 5. Julia sets: (a) $f(z) = z^2$, (b) $f(z) = z^2 + (-0.2, 0.6)$, (c) Douady's rabbit, (d) $f(z) = z^2 + i$, (e) $f(z) = z^2 + Lz$, L = eighth root of unity, and (f) $f(z) = z^2 + 0.3$.

under iteration of $z^2 + \lambda$. If $|f_n(0)|$ is sufficiently large, say larger than 2, then $f_n(0)$ will converge to ∞. This indicates that the Julia set for that value of λ is not a simple closed curve. However, if $|f_n(0)|$ remains bounded, then the Julia set is a simple closed curve.

2.2 Algorithms

The programs in listing 1 allow the user to plot some pointillistic fractal images called Julia sets (listings are provided at the end of this section). They work for all values of λ and any starting point. They require only a simple point plotter to produce images, and only as much memory as the number of iterates desired. The programs were written in VAX-11 FORTRAN (DEC) because it supports a complex variables format and has many intrinsic math functions. If FORTRAN is not available, the code listings can serve as a model for writing a program.*

(For the reader who has braved the theory section, an explanation of why these algorithms work is now simple. First, the Julia set is the set in which the function does not have convergent neighborhoods. It is preserved under backward iteration, $(f^{-1})_n$. For example, if

$$w = f(z) = z^2 + \lambda \quad ,$$

$$z = f^{-1}(w) = \pm\sqrt{w - \lambda} \quad ,$$

and so

$$(f^{-1})_n = f^{-1} \circ \ldots \circ f^{-1} \quad , \text{ n times.}$$

**Without the complex variable format, support routines have to be written. For example, multiplication must follow the rule*

$$(a,b) * (c,d) = (ac - bd, ad + bc) \quad .$$

A complex square root can be written using polar coordinates. Since

$$\sqrt{z} = \sqrt{r e^{it}} = \sqrt{r}\ \sqrt{e^{it}} \quad ,$$

by an application of Euler's formula and the half-angle formulas, we obtain

$$\sqrt{z}\ \sqrt{r}\left[\pm\left(\frac{1+\cos t}{2}\right)^{1/2} \pm i\left(\frac{1-\cos t}{2}\right)^{1/2}\right] \, .$$

where signs are chosen according to quadrant. Therefore, because

$$|z| = r = (x^2 + y^2)^{1/2} \ , \ x = r\cos t \ , \text{ and } \ y = r \sin t \ ,$$

we obtain

$$(x,y)^{1/2} = \left(\pm\left[\frac{r+x}{2}\right]^{1/2} \ , \ \pm\left[\frac{r-x}{2}\right]^{1/2}\right) \, .$$

Therefore, to get an iterate into the Julia set, iterate backwards, choosing a branch (±) of the square root at random. Thus, the iterate is never allowed to converge, and therefore must land in the Julia set. Once captured there, it just moves around within the set.)

Color dynamical images in $\hat{C}$ are produced by a different procedure (see fig. 6, center spread, pp 22 and 23). Here, the colors represent convergence rates. After a domain has been chosen, the image is produced by proceeding pixel by pixel across that domain, iterating the function for the value of z represented by that pixel. Iteration continues until $|f_n(z)|$ reaches a certain size, or the function has been iterated a predetermined maximum number of times.

The number of iterations then determines the color of that pixel. (This is essentially the same procedure which was outlined by Dewdney, 1985.) It is interesting to note that in producing these color images, it is possible to see the various orbits of the regions computed.

The color images in parameter space are produced in the same way as described in the procedure above. However, in this situation, a given value of z is fixed throughout the entire procedure, while the position of the pixel determines the value of λ.

The color images of both the $\hat{C}$ dynamics and the parameter spaces are not limited to iterating $f(z) = z^2 + \lambda$. All the color images shown in figure 6 came from a different dynamic.

Listing 2 is a FORTRAN program for iterating Newton's method of finding zeroes as applied to $f(z) = z^7 - 1$. This produced the pixel information to create figure 6(i).

There were a few tricks involved in computing and storing the data files for these color images. As this type of program is computationally intensive and requires quite a bit of computer time, computations were simplified when possible (e.g., using $|z|^2$ instead of $|z|$, thus eliminating a square root for each iteration). Since the resulting data file would occupy much disk space, only the minimal amount of information needed to produce an image was stored. Pixels were computed sequentially on a given row with the function producing a color value for each pixel. Adjacent pixels of the same color were considered a horizontal vector. When a new color value for a pixel was computed, the previous pixel's column position (terminal point for the vector) and the vector's color were loaded into a large data buffer:

$$\begin{gathered} \text{databuf}(n) \leftarrow x \\ \text{databuf}(n+1) \leftarrow \text{oldcount} \\ n \leftarrow n + 2 \\ \text{oldcount} \leftarrow \text{count} \\ X \leftarrow X + 1 \end{gathered}$$

When the buffer became full, it was written to a disk file as a block of unformatted data. A second program (listing 3) must be used to plot the vector file. Other techniques that are hardware dependent were also used.

Listing 1. Code producing data for Julia set images

```
c ------------------------------------------------------------------------------
c  Plots Julia sets for quadratic maps by iterating
c                   w = SQRT(z - c).
c  Some subroutine calls are intrinsic to VAX-11 FORTRAN (DEC)
c  Programmer : S. Casey
c ------------------------------------------------------------------------------

      PROGRAM FRAC
      COMPLEX*16 z, c, CDSQRT
      REAL*8 x, y
      INTEGER*4 Niter, Iseed, Timeseed

      WRITE (6,*) ' This routine plots Julia sets for f(z) = z**2 + c.'
      WRITE (6,*) ' Enter the constant ... c=(a,b)'
      READ (5,*) c
      WRITE (6,*) ' Enter the initial z value ... z=(a,b)'
      READ (5,*) z
      WRITE (6,*) ' Enter the # of backward iterates ... Niter'
      READ (5,*) Niter

      Iseed = Timeseed()   ! get seed for random number generator
c     RAN(Iseed) returns a floating-point number >= 0.0 and < 1.0

      OPEN ( 10, FILE='FRAC.DAT', STATUS='NEW', ERR=999, IOSTAT=IOS,
     1   CARRIAGECONTROL='LIST' )

      DO I = 1, Niter    !  Plots positive branch
         IF ( RAN(Iseed) .LT. 0.5 ) THEN
            z = -1. * z
         ENDIF
         z = z - c
         z = CDSQRT(z)
         x = DREAL(z)
         y = DIMAG(z)
         IF ( I .GE. 11 ) THEN    !  Let fn(z) converge into Julia set
            WRITE(10,*) x, y
         ENDIF
      ENDDO

      DO I = 1, Niter    !  Other branch
         IF ( RAN(Iseed) .LT. 0.5 ) THEN
            z = -1. * z
         ENDIF
         z = z - c
         z = -1. * CDSQRT(z)
         x = DREAL(z)
         y = DIMAG(z)
         IF ( I .GE. 11 ) THEN    !  Let fn(z) converge into Julia set
            WRITE(10,*) x, y
         ENDIF
      ENDDO

      CLOSE(10)
      STOP
999   WRITE (6,*) ' Error opening new file FRAC.DAT - ', IOS
      STOP
      END
```

```
c -----------------------------------------------------------------------

      INTEGER*4 FUNCTION Timeseed ()
c     This function returns a large, odd integer to serve as an initial
c      seed for a random number generator.

      Timeseed = INT(SECNDS(0.0))  ! get number of seconds since midnight
      IF ( MOD(Timeseed,2) .EQ. 0 ) Timeseed = Timeseed + 1  ! odd value
      RETURN
      END


c -----------------------------------------------------------------------
c  Plots Julia sets for f(z) = z**2 + L*z by iterating
c           1/2( -L (+/-) (L**2 + 4*z) ).
c  Some subroutine calls are intrinsic to VAX-11 FORTRAN (DEC)
c  Programmer : S. Casey
c -----------------------------------------------------------------------

      PROGRAM FRAC2
      COMPLEX*16 z, L, Det, Rootunity, CDSQRT
      REAL*8 x, y
      INTEGER*4 K, Niter, Iseed, Timeseed
      CHARACTER Answer

      WRITE (6,*) ' This routine plots Julia sets for f(z) = z**2 + L*z.'
      WRITE (6,*) ' The results are interesting if'
      WRITE (6,*) ' L is a root of unity.'
      WRITE (6,*) ' Do you wish to enter a root of unity?'
      WRITE (6,*) ' Answer y for yes.'
95    FORMAT (A)
      READ  (5,95) Answer

      IF ((Answer .EQ. 'Y') .OR. (Answer .EQ. 'y')) THEN
         WRITE (6,*) ' Enter the integer denominator.'
         READ (5,*) K
         L = Rootunity (K)
         WRITE (6,*) ' L = ', L
         GO TO 20
      ENDIF

      WRITE (6,*) ' Enter the constant ... L=(a,b)'
      READ (5,*) L
20    WRITE (6,*) ' Enter the initial z value ... z=(a,b)'
      READ (5,*) z
      WRITE (6,*) ' Enter the # of backward iterates ... Niter'
      READ (5,*) Niter

      Iseed = Timeseed()   ! get seed for random number generator
c     RAN(Iseed) returns a floating-point number >= 0.0 and < 1.0

      OPEN ( 10, FILE='FRAC2.DAT', STATUS='NEW', ERR=999, IOSTAT=IOS,
     *   CARRIAGECONTROL='LIST' )
```

Listing 1. Code producing data for Julia set images (cont'd)

```
      DO I = 1, Niter
         DET = CDSQRT ( (L * L) + (4. * z) )
         IF ( RAN(Iseed) .LT. 0.5 ) THEN
            DET = -1. * DET
         ENDIF
         z = (1./2.) * ( (-1.* L) + DET )
         x = DREAL(z)
         y = DIMAG(z)
         IF ( I .GE. 11 ) THEN    !  Let fn(z) converge into Julia set
            WRITE(10,*) x, y
         ENDIF
      ENDDO

      CLOSE(10)
      STOP
999   WRITE (6,*) ' Error opening new file FRAC2.DAT = ', IOS
      STOP
      END

c -----------------------------------------------------------------------

      INTEGER*4 FUNCTION Timeseed ()
c     This function returns a large, odd integer to serve as an initial
c      seed for a random number generator.

      Timeseed = INT(SECNDS(0.0))   ! get number of seconds since midnight
      IF ( MOD(Timeseed,2) .EQ. 0 ) Timeseed = Timeseed + 1  ! odd value
      RETURN
      END

c -----------------------------------------------------------------------

      COMPLEX*16 FUNCTION Rootunity(K)

      COMPLEX*16 CDEXP, ITPIK
      REAL*8 PI, TPIK
      INTEGER*4 K

      PI = 3.14159265358979323846
      TPIK = (2. * PI) / K
      ITPIK = (0.0,1.0) * TPIK
      Rootunity = CDEXP( ITPIK )
      RETURN
      END
```

Listing 2. Code producing pixel information for iteration of Newton's method as applied to $f(z) = z^7 - 1$

```
c -----------------------------------------------------------------------
c  Iteration of Newton's method on f(z) = z**7 - 1.
c  Some subroutine calls are intrinsic to VAX-11 FORTRAN (DEC)
c  Programmers: R. Miller and S. Casey
c -----------------------------------------------------------------------

      PROGRAM NEWT
      COMPLEX*16 Znew, Zold, Ztemp
      REAL*8 Diffabs
      INTEGER*2 iterations, oldcount, X, count, rows, cols, M, N
      INTEGER*2 databuf(16384)
      INTEGER*4 K, J
      REAL*8 acorner, bcorner, side, gap, realc, imagc

      COMMON /BLOCK1/ databuf, K

c Parms passed from command line: iterations, acorner, bcorner, side, cols
c -----------------------------------------------------------------------
c  This routine calculates pixel information for a rectangular region R.
c    iterations = maximum number of times calculating loop is executed
c    acorner = Real part of coordinate of lower left hand corner of R
c    bcorner = Imaginary part of coordinate of lower left hand corner of R
c    side = length of horizontal side of R
c    rows,cols = number of pixels in R
c -----------------------------------------------------------------------
      ACCEPT *, iterations, acorner, bcorner, side, cols
      rows = cols

      OPEN( 10, FILE='NEWT.VEC', STATUS='NEW', IOSTAT=IOS,
     * ERR=999, FORM='UNFORMATTED' )

      CALL OUTBUF( cols, iterations )        ! load first data pair

      gap = side / REAL(rows)
      DO N = rows - 1, 0, -1                 ! compute pixels
         imagc = REAL(N) * gap + bcorner
         DO M = 0, cols - 1
            realc = REAL(M) * gap + acorner
            Znew = DCMPLX( realc, imagc )    ! combine real/imaginary parts
            count = 0
            Diffabs = 1.0

            DO WHILE((count .LT. iterations).AND.(Diffabs .GT. 0.0001))
               Zold = Znew
               Ztemp = 7.0 * Znew**6
               IF ( CDABS(Ztemp) .LT. .00001 ) THEN    ! absolute value
                  count = 0
                  GOTO 222
               ENDIF
               Znew = ((6.0 * Znew**7) + 1.0) / Ztemp
               Diffabs = CDABS( Znew - Zold )
               count = count + 1
            ENDDO
```

Listing 2. Code producing pixel information for iteration of Newton's method as applied to $f(z) = z^7 - 1$ (cont'd)

```
c     Load vector data: column X or M and color 'count' or 'oldcount'

222         IF( M .EQ. 0 ) THEN
               oldcount = count
               X = 0
            ELSE IF( M .EQ. (cols - 1) ) THEN
               IF( count .NE. oldcount ) CALL OUTBUF( X, oldcount )
               CALL OUTBUF( M, count )
            ELSE
               IF( count .NE. oldcount ) THEN
                  CALL OUTBUF( X, oldcount )
                  oldcount = count
               ENDIF
               X = X + 1
            ENDIF

         ENDDO
      ENDDO

      IF(( K .GT. 0 ) .AND. (K .LT. 16384)) THEN
         DO J = K+1, 16384
            databuf(J) = 0! fill remainder of buffer
         ENDDO
         WRITE(10) databuf! write last record
      ENDIF

      CLOSE(10)
      CALL EXIT

999   WRITE (6,*) ' Error opening new file NEWT.VEC ', IOS
      CALL EXIT
      END

c -----------------------------------------------------------------------

      SUBROUTINE OUTBUF( X, color )
      INTEGER*2 X, color

      INTEGER*2 databuf(16384)
      INTEGER*4 K/0/! data buffer index

      COMMON /BLOCK1/ databuf, K

      K = K + 1
      databuf(K) = X
      K = K + 1
      databuf(K) = color

      IF( K .EQ. 16384 ) THEN
         WRITE(10) databuf             ! write vector data
         K = 0
      ENDIF

      RETURN
      END
```

Listing 3. Code/pseudocode for plotting color fractal images

```
      PROGRAM PLOTFRAC
c ---------------------------------------------------------------------------
c     Pseudocode/FORTRAN listing illustrating how to plot vector files.
c     The variable 'iterations' can be used for loading color look-up table
c     Square images are produced ( rows = cols ).
c     Programmer : R. Miller
c ---------------------------------------------------------------------------

      INTEGER*2 iterations, X, Y, color, rows, cols
      INTEGER*2 xcenter, ycenter
      CHARACTER filename*40

      filename = 'NEWT.VEC'
      OPEN( 10, FILE=filename, STATUS='OLD', FORM='UNFORMATTED' )

      CALL READBUF(cols, iterations)  ! get first data pair
      rows = cols

c     allocate I/O device

c     enter graphics mode
c     reset graphics device

      xcenter = cols / 2
      ycenter = rows / 2
c     set screen and coordinate origins to center image on the screen

c     load 'iterations' number of colors into look-up tables

      Y = rows - 1                    ! range of rows for plot: 0 to rows -
      cols = cols - 1                 ! range of columns      : 0 to cols -

c     plot left to right and top to bottom of screen
      DO WHILE( Y .GE. 0 )

c        MOVE 0,Y                     ! Move to beginning of row Y
         CALL READBUF(X, color)       ! get first vector data pair for row
         DO WHILE( X .LT. cols )

c           VALUE color               ! set current drawing color
c           DRAW X+1,Y                ! draw to X+1 to avoid a MOVE X+1,Y
            CALL READBUF(X, color)    ! get next vector data pair

         ENDDO
c        VALUE color                  ! set drawing color
c        DRAW X,Y                     ! draw last vector of row Y
         Y = Y - 1

      ENDDO

c     exit graphics mode
c     de-allocate I/O device
      CLOSE(10)                       ! close file

      CALL EXIT
      END
```

Listing 3. Code/pseudocode for plotting color fractal images (cont'd)

```
c ----------------------------------------------------------------------

      SUBROUTINE READBUF( X, color )
      INTEGER*2 X, color
      INTEGER*2 databuf(16384)
      INTEGER*4 K/0/                      ! data buffer index

      IF( K .EQ. 0 ) READ(10) databuf

      K = K + 1
      X = databuf(K)
      K = K + 1
      color = databuf(K)

      IF( K .EQ. 16384 ) K = 0

      RETURN
      END
```

3. SEED FRACTALS

Fractal images may also be generated by the repetition of a given geometric pattern. Examples of these types of fractals are seen in figures 1 to 4. These images are generally called seed fractals.

By one definition, a fractal curve is a curve for which the fractional dimension exceeds the topological dimension. Unless the curve is a space-filling Peano curve, this topological dimension is 1. In some instances, the curve may be a simple closed curve, as in the case of Koch's snowflake. If the fractal is a seed fractal, the curve is constructed by a limit procedure where a given seed design (some geometric shape) is scaled and repeated. This produces a curve that is self-similar; that is, the curve pattern repeats itself on any level of magnification.

3.1 Procedure to Produce Seed Fractal Images

All the curves in figures 1 to 4 are seed fractals. In each of these, the algorithm to produce them was a twofold process. The basic pattern had to be calculated, scaled, and moved to its proper place via a similarity transformation. Simultaneously, a data structure had to be set up in order to prepare for the next level of iteration. The first step was achieved through trigonometry and linear algebra. The second step was handled through counting.

Listings 4 and 5 are the code listings for Sierpinski's gasket (fig. 7) and the Cantor-Lebesgue function (fig. 2). The first of these demonstrates the drawing process, while the second provides an example of the counting process. (Both listings are given at the end of this section, pp 28 to 31.

3.2 Calculation of Topological and Fractional Dimension

Seed fractals present a good opportunity to demonstrate the calculation of fractional dimension, as is seen in the following examples. Fractional dimension is a precise gauge on how much an object "wiggles about," and is given by a real number. It is different from our more intuitive understanding of dimension, which is expressed precisely by topological dimension. (Background for this section is provided by Guckenheimer and Holmes (1983), Hurewicz and Wallman (1948), Lehto and Virtanen (1973), and Munkres (1973).)

Intuitively, given a mathematical object in Euclidean 3-space, that object is usually thought of as having dimension 0, 1, 2, or 3 (which are the dimensions of a point, line segment, square, and cube, respectively). This intuitive dimension is topological dimension. Topological dimension is always given by an integer, and corresponds to the minimal integer value, say m, for which the following holds: given a topological space X, and an open cover A of that space, there is a refinement B of A that has order m + 1. Here, a collection B of subsets of A has order m + 1, if some point of A lies in m + 1 elements of B, and no point of A lies in more than m + 1 elements of B.

Let the diameter of a set be the least upper bound, or supremum (sup) of the set of distances between points in the set.

Example: The unit interval I = [0,1] has topological dimension 1.

Let I be endowed with the subspace topology inherited from the Euclidean metric topology on R.

To prove that dim I $\neq$ 0, assume that the unit interval is connected, that is, does not have two disjoint nonempty subsets whose union equals I. For $0 < \varepsilon < 1$, let A be any open covering of sets of diameter less than ε. Suppose that A has order 1. Then, no two elements of A intersect. Also, since $\varepsilon < 1$, A must contain at least two elements. Let U be one element of A, and let V be the union of the others. Then $U \cap V = \emptyset$ and $U \cup V = I$, contradicting connectedness.

Next, let A be any open cover of I. Then,

$$A = \{[0,a_2), (a_1,a_3), \ldots, (a_{n-2},1]\}$$

for some partition P_1 of the unit interval.

$$0 = a_0 < a_1 < \ldots < a_{n-1} < a_n = 1 \quad .$$

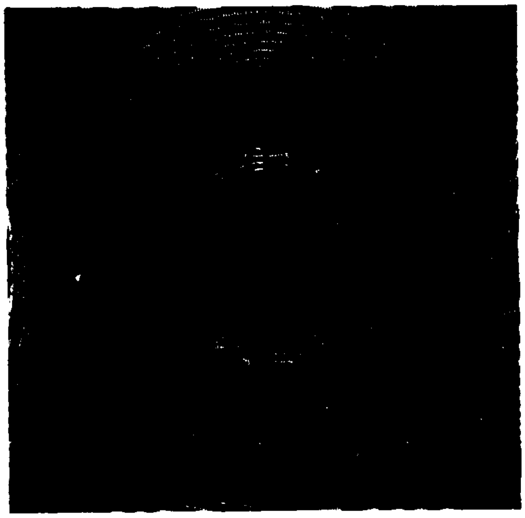
Let $\varepsilon = \min_i \{|a_i - a_{i-1}|\}$. Now, refine P_1 to a partition P_2 so that

$$P_2 = \{b_i: 0 = b_0 < b_1 < \ldots < b_{m-1} < b_m = 1\}$$

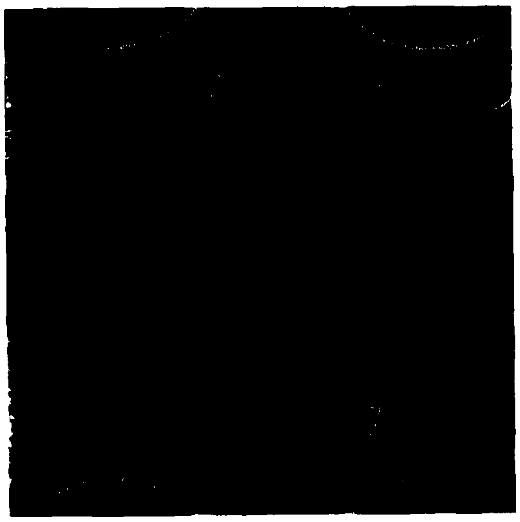
with $|b_i - b_{i-1}| < \varepsilon/2$ for all i.

(a)

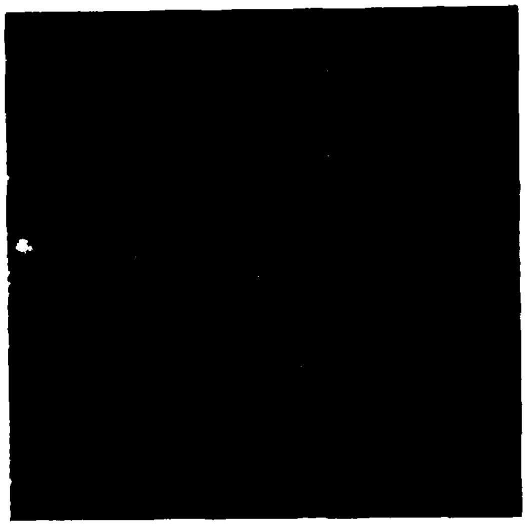
(b)



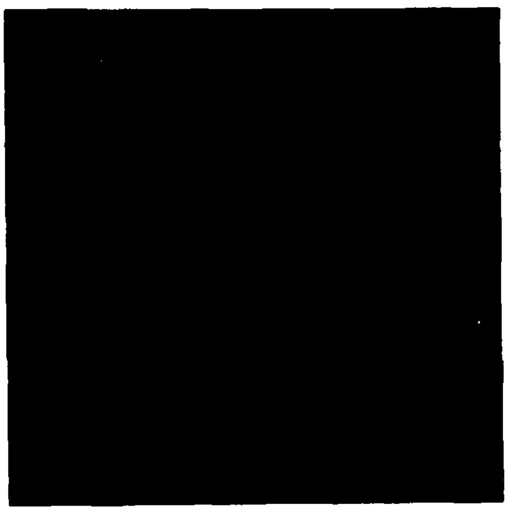
(c)



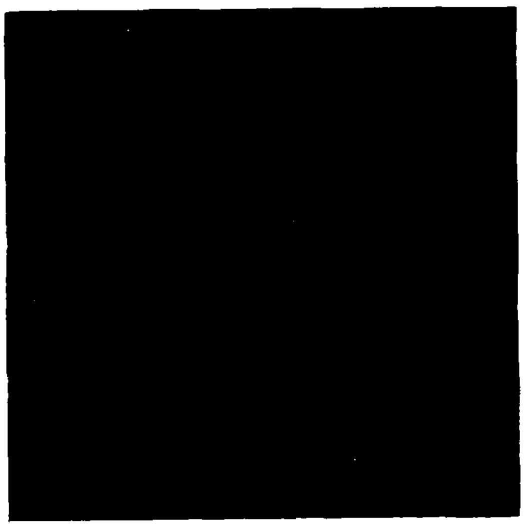
(d)



(e)



(f)



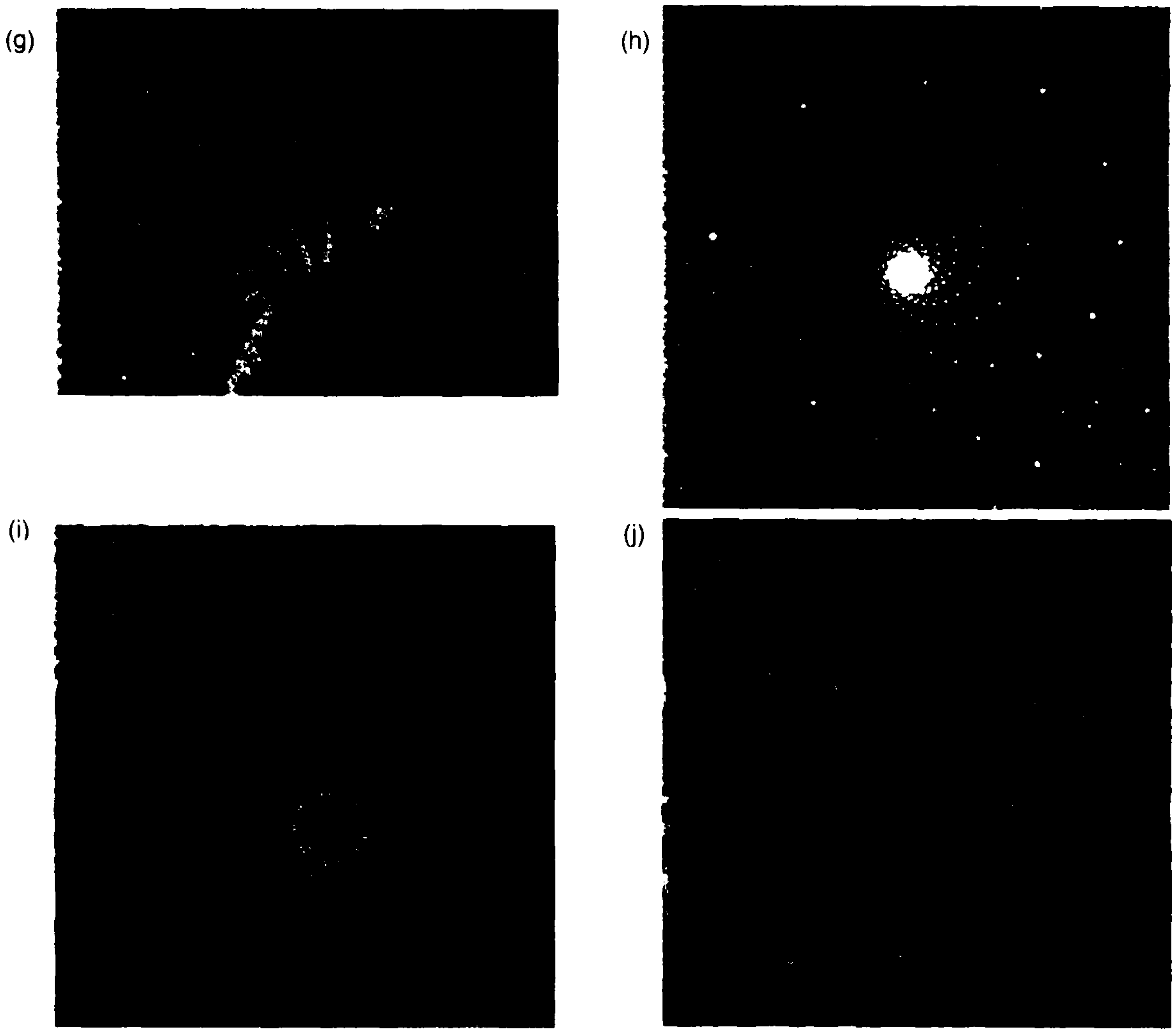


Figure 6. Color images from dynamics in $\hat{C}$: (a) $f(z) = \lambda \cdot \exp(z)$, parameter space; (b) $f(z) = \lambda \cdot \exp(z^2)$, parameter space (note symmetry, as expected); (c) $f(z) = \lambda \cdot \sin(z)$, parameter space (note "mini-Mandelbrot sets"); (d) $f(z) = \lambda \cdot \cos(z)$, parameter space; (e) $f(z) = \lambda \cdot \tan(z)$, parameter space; (f) $f(z) = \lambda \cdot \tan(z)$, parameter space (enlargement of a region in (e)); (g) $f(z) = z^4 - z - \lambda$, parameter space; (h) $f(z) = z^4 - z - \lambda$, parameter space (enlargement of a region in (g)); (i) Newton's method applied, $f(z) = z^7 - 1$, C-dynamical system; and (j) $f(z) = z^2 + \lambda$, parameter space (enlargement of a region on the boundary of the Mandelbrot set, containing an outline of the original set).

Pixel colors represent convergence rates. In all the figures above, this rate was an "escape rate"--how quickly the iterates of that pixel converged to infinity. In general, the color schemes went according to color frequency. Thus, red was slow convergence, and violet was fast convergence. However, this scheme was not strictly adhered to. Also, scaling was required, especially in figures (a) to (f) (expotential growth $\rightarrow$ logarithmic scaling).

All the color images seen were produced by R. Miller, who worked on computer graphics, and S. Casey, who did mathematical programming. The images were produced in the spring and summer of 1986.

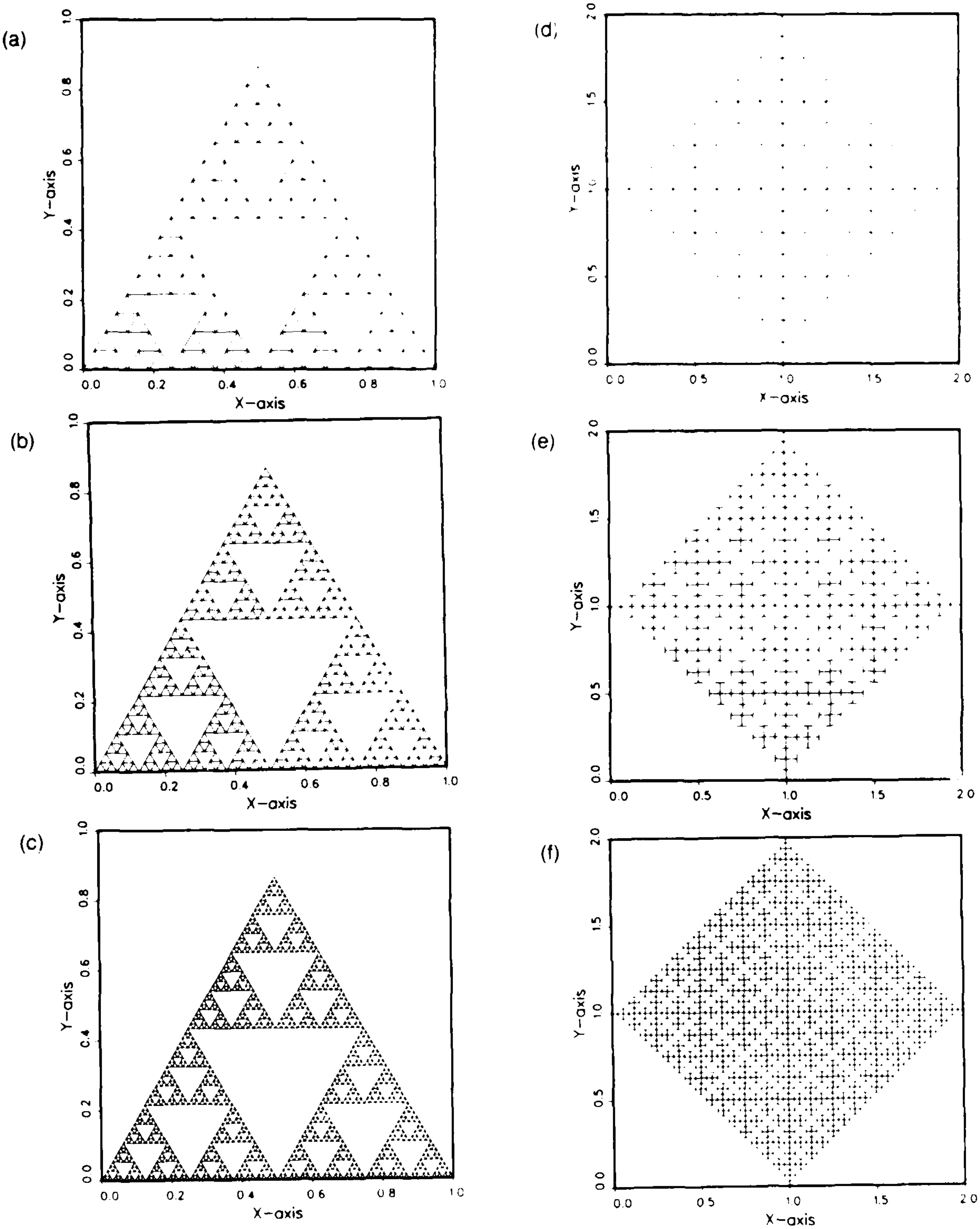


Figure 7. Sierpinski's gasket (a-c) and the antenna (d-f): (a) defining procedure followed three times, (b) four times, (c) five times; and (d) defining procedure followed three times, (e) four times, (f) five times.

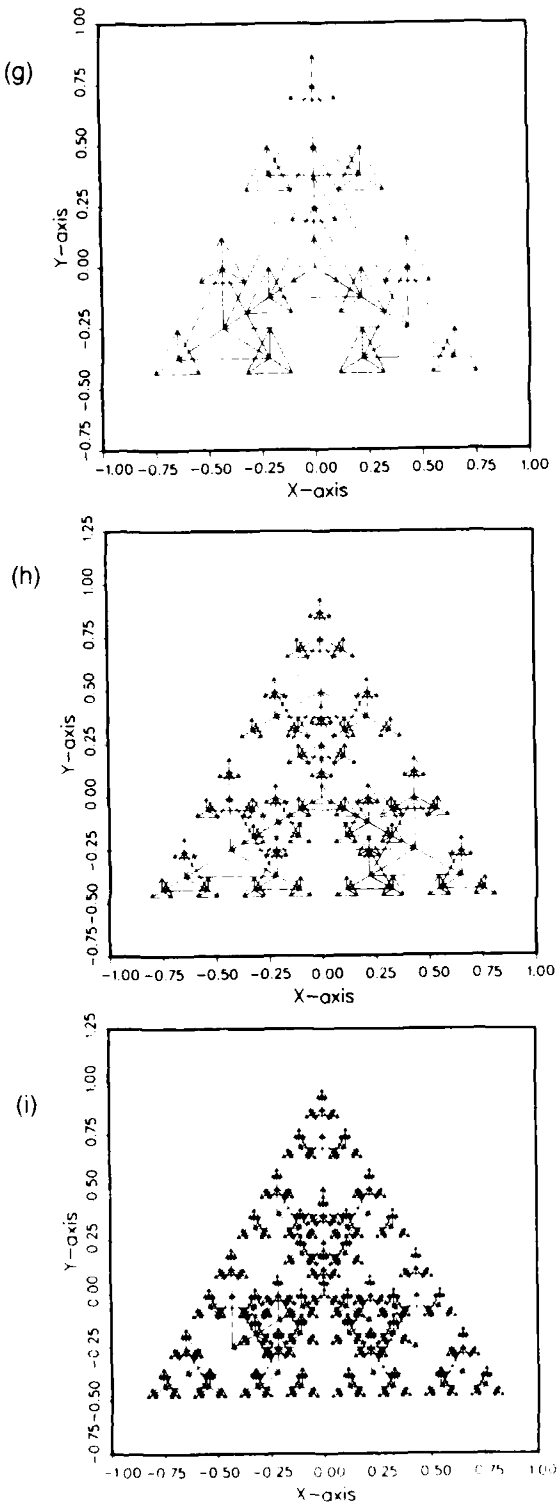


Figure 7 (cont'd). The arrowhead: (g) defining procedure followed three times, (h) four times, and (i) five times.

Then,

$$B = \{[0,b_2), (b_1,b_3), \ldots , (b_{m-2},1]\}$$

is a refinement of A of order 2. Since this is true for any cover A, it follows that

$$\dim I \leq 1 \quad .$$

Therefore, since topological dimension is integer valued, dim I = 1. ◇

To prove that the topological dimension of a simple (nonintersecting) continuous curve is equal to 1 requires a bit more machinery.

In order to calculate fractional dimension, the idea of covering a set by open sets is used. However, in this situation, the diameter of the sets used plays an important role. The idea of fractional dimension goes back to the nineteenth century, to F. Hausdorff and I. Besocovitch. The following is the formal definition of the fra'tional dimension named after them.

Let X be some set in Euclidean 3-space. Consider all coverings of X by countably many sets of diameter less than some $d > 0$. Given one such covering, say $\{\alpha_1, \alpha_2 \ldots\}$, associate with that cover the number

$$\sum_n d_n^{\beta} \quad ,$$

where d_n is the diameter of the set α_n and $\beta > 0$. For every β let

$$\mu_\beta^*(X) = \lim_{d \to 0} \left[\inf \left(\sum_n d_n^{\beta}\right)\right] \quad ,$$

where inf (infimum) is the greatest lower bound. The quantity μ_β^* is called the β-dimensional Hausdorff-

Bes covitch outer measure. Then, the Hausdorff-Besocovitch dimension of X is given by

$$\dim X = \inf \{\beta \mid \mu_\beta^*(X) = 0\} \quad .$$

Again, this definition is best seen in the following examples.

Example--the Cantor middle thirds set: Let $C_0 = [0, 1]$. Then,

$$C_1 = C_0 - \left(\frac{1}{3}, \frac{2}{3}\right) ,$$

$$C_2 = C_1 - \left[\left(\frac{1}{9}, \frac{2}{9}\right), \left(\frac{7}{9}, \frac{8}{9}\right)\right] ,$$

...

$$C_{n+1} = C_n - \left[\left(\frac{1}{3^{n+1}}, \frac{2}{3^{n+1}}\right), \ldots, \left(\frac{3^{n+1}-2}{3^{n+1}}, \frac{3^{n+1}-1}{3^{n+1}}\right)\right] \quad .$$

The Cantor set C is $\lim_{n\to\infty} C_n$. Note that the set $[0, 1] - C$ has length

$$\sum_{n=0}^{\infty} \left(\frac{2^n}{3^{n+1}}\right) = \frac{1}{3} \sum_{n=0}^{\infty} \left(\frac{2}{3}\right)^n = \frac{1}{3}\left(\frac{1}{1-(2/3)}\right) = 1 \quad .$$

To get an upper bound on dim C, first work with the set C_n. This set can be covered by 2^n sets of length 3^{-n}. Now,

$$\sum_n d_n^\beta = 2^n (3^{-n})^\beta \quad ,$$

and so (nonrigorously)

$$\mu_\beta^*(C) = \liminf_{n\to\infty} 2^n (3^{-n})^\beta \quad .$$

To calculate the dimension of C, one must find the $\inf \{\beta \mid \mu_\beta^*(C) = 0\}$. As

$$2^n 3^{-n\beta} = x \leftrightarrow n(\log 2 - \beta \log 3) = \log x \quad ,$$

and

$$\log x \to \infty \quad \text{as} \quad n \to \infty \leftrightarrow \beta < \frac{\log 2}{\log 3} \quad ,$$

we obtain

$$\dim C \leq \frac{\log 2}{\log 3} \quad .$$

With a bit more work, one can see that this bound is sharp. ◇

Example--Koch's snowflake: At the n^{th} stage of its construction, the snowflake can be covered by 4^n sections of length 3^{-n}. Thus,

$$\mu_\beta^*(K) = \liminf_{n\to\infty} 4^n 3^{-n\beta} \quad .$$

Therefore, since

$$4^n e^{-n\beta} = x \leftrightarrow n(\log 4 - \beta \log 3) = \log x \quad ,$$

and

$$\log x \to \infty \text{ as } n \to \infty \leftrightarrow \beta < \frac{\log 4}{\log 3} \, ,$$

we obtain

$$\dim K \leq \frac{\log 4}{\log 3} \quad . \quad \diamond$$

Discussions of topological dimension may be found in Munkres (1975) and Hurewicz and Wallman (1948). Discussions of fractional dimension may be found in Lehto and Virtanen (1973) and Mandelbrot (1983). One of Mandelbrot's main underlying themes is that a fractal is characterized by its fractional dimension. Thus, numerous calculated examples can be found throughout the book.

Listing 4. Code producing vector data to draw Sierpinski's gasket

```
c ----------------------------------------------------------------------------
c  This routine draws Sierpinski's Gasket.
c  Some subroutine calls are intrinsic to VAX-11 FORTRAN (DEC)
c  Programmer : S. Casey
c ----------------------------------------------------------------------------

      PROGRAM SGASKET
      REAL*8 Con, Scale
      REAL*8 Xnode(7,730), Ynode(7,730)  ! Centers of the triangles...
      REAL*8 Xreturn(3), Yreturn(3)
      INTEGER Iter, I, J, K
      CHARACTER*1 Lntyp   ! Draws either points(P) or vectors(V)...

      WRITE (6,*) ' Enter the # of iterates ... a maximum of 7.'
      READ (5,*) Iter

      OPEN (10, FILE='SGASKET.DAT', STATUS='NEW', ERR=999, IOSTAT=IOS,
     *   CARRIAGECONTROL='LIST' )

      Con = 3.0
      Con = DSQRT(Con)

 100  FORMAT(1X,F10.3,1X,F10.3,1X,A)
      Lntyp = 'P'
      WRITE(10,100) 0.0, 1.0, Lntyp  ! Scaling factors.
      WRITE(10,100) 0.0, 0.0, Lntyp
      Lntyp = 'V'
      WRITE(10,100) 0.0, 0.0, Lntyp  ! The outside triangle.
      WRITE(10,100) 1.0, 0.0, Lntyp
      WRITE(10,100) 0.5, (Con / 2.0), Lntyp
      WRITE(10,100) 0.0, 0.0, Lntyp

      Lntyp = 'P'    ! Lifting the pen.
      WRITE(10,100) 0.25, (Con / 4.0), Lntyp
      Lntyp = 'V'
      WRITE(10,100) 0.75, (Con / 4.0), Lntyp   ! The inside triangle.
      WRITE(10,100) 0.5, 0.0, Lntyp
      WRITE(10,100) 0.25, (Con / 4.0), Lntyp

      Xnode(1,1) = 0.5
      Ynode(1,1) = Con / 8.0

      DO I = 1, Iter
         Scale = 1.0 / (2.0**(I+1))
         DO J = 1, 3**(I-1)

            CALL Gasketseed(Xnode(I,J), Ynode(I,J), Scale,
     *               Xreturn, Yreturn)

            DO K = 1, 3
               Xnode((I+1), (3*J-(3-K))) = Xreturn(K)
               Ynode((I+1), (3*J-(3-K))) = Yreturn(K)
            ENDDO
         ENDDO
      ENDDO

      CLOSE(10)
      STOP
999   WRITE (6,*) ' Error opening new file SGASKET.DAT - ', IOS
      STOP
      END
```

Listing 4. Code producing vector data to draw Sierpinski's gasket (cont'd)

```
c ----------------------------------------------------------------------

      SUBROUTINE Gasketseed(Xnode, Ynode, Scale, Xreturn, Yreturn)

      REAL*8 Xnode, Ynode, Scale, Con
      REAL*8 Xreturn(3), Yreturn(3)
      REAL*8 Xvar(3,4), Yvar(3,4)
      CHARACTER*1 Lntyp   ! Draws either points(P) or vectors(V)...

 100  FORMAT(1X,F10.3,1X,F10.3,1X,A)

      Con = 3.0
      Con = DSQRT(Con)

      Xreturn(1) = 1.0 / 2.0
      Yreturn(1) = -Con / 8.0

      Xreturn(2) = 0.0
      Yreturn(2) = (3.0 * Con) / 8.0

      Xreturn(3) = -1.0 / 2.0
      Yreturn(3) = -Con / 8.0

      DO I = 1, 3
         Xreturn(I) = (2.0 * Scale * Xreturn(I)) + Xnode
         Yreturn(I) = (2.0 * Scale * Yreturn(I)) + Ynode
      ENDDO

      DO I = 1, 3

         Xvar(I,1) = -1.0 / 2.0
         Yvar(I,1) = Con / 4.0

         Xvar(I,2) = 1.0 / 2.0
         Yvar(I,2) = Con / 4.0

         Xvar(I,3) = 0.0
         Yvar(I,3) = -Con / 4.0

         Xvar(I,4) = Xvar(I,1)
         Yvar(I,4) = Yvar(I,1)

         Lntyp = 'P'
         DO J = 1, 4
            Xvar(I,J) = (Scale * Xvar(I,J)) + Xreturn(I)
            Yvar(I,J) = (Scale * Yvar(I,J)) + Yreturn(I)
            IF ( J .EQ. 2 ) Lntyp = 'V'
            WRITE(10,100) Xvar(I,J), Yvar(I,J), Lntyp
         ENDDO

      ENDDO

      RETURN
      END
```

Listing 5. Code producing vector data to draw Cantor-Lebesgue function

```
c ------------------------------------------------------------------------
c  This routine draws the Cantor-Lebesgue funtion.
c  Some subroutine calls are intrinsic to VAX-11 FORTRAN (DEC)
c  Programmer : S. Casey
c ------------------------------------------------------------------------

      PROGRAM CANTOR
      REAL*8 X1(8,256), Y1(8,256), X2(8,256), Y2(8,256), Xstep, Ystep
      REAL*8 Xreturn/1.0/, Yreturn/0.0/, Xscale1/0.0/, Yscale1/0.0/
      REAL*8 Xscale2/1.0/, Yscale2/1.0/
      INTEGER*4 Iter, Count/0/

      WRITE (6,*) ' Enter the # of iterates ... a maximum of 8.'
      READ (5,*) Iter

      OPEN ( 10, FILE='CANTOR.DAT', STATUS='NEW', ERR=999, IOSTAT=IOS,
     1   CARRIAGECONTROL='LIST' )

      DO I = 1, Iter
         IF (I .EQ. Iter) THEN
            WRITE(10,*) Xscale1, Yscale1
         END IF
         Count = Count + 2**(I-1)

         DO J = 1, Count
            Xstep = (1.0 / (3.0**I))
            Ystep = (1.0 / (2.0**I))
            IF (J .EQ. 1) THEN
               X1(I+1,J) = Xstep
               Y1(I+1,J) = Ystep
               X2(I+1,J) = X1(I+1,J) + Xstep
               Y2(I+1,J) = Y1(I+1,J)
               IF (I .EQ. Iter) THEN
                  WRITE(10,*) X1(I+1,J), Y1(I+1,J)
                  WRITE(10,*) X2(I+1,J), Y2(I+1,J)
               END IF
            END IF

            IF (MOD(J,2) .EQ. 0) THEN
               X1(I+1,J) = X1(I,(J/2))
               Y1(I+1,J) = Y1(I,(J/2))
               X2(I+1,J) = X2(I,(J/2))
               Y2(I+1,J) = Y2(I,(J/2))
               IF (I .EQ. Iter) THEN
                  WRITE(10,*) X1(I+1,J), Y1(I+1,J)
                  WRITE(10,*) X2(I+1,J), Y2(I+1,J)
               END IF
            ELSE
               X1(I+1,J) = X2(I,(J/2)) + Xstep
               Y1(I+1,J) = Y2(I,(J/2)) + Ystep
               X2(I+1,J) = X1(I+1,J) + Xstep
               Y2(I+1,J) = Y1(I+1,J)
               IF (I .EQ. Iter) THEN
                  WRITE(10,*) X1(I+1,J), Y1(I+1,J)
                  WRITE(10,*) X2(I+1,J), Y2(I+1,J)
               END IF
            END IF
         ENDDO
```

Listing 5. Code producing vector data to draw Cantor-Lebesgue function (cont'd)

```
            IF (I .EQ. Iter) THEN
               WRITE(10,*) Xscale2, Yscale2
               WRITE(10,*) Xreturn, Yreturn
            END IF
         ENDDO

         CLOSE(10)
         STOP
999      WRITE (6,*) ' Error opening new file CANTOR.DAT - ', IOS
         STOP
         END
```

4. SQUIG FRACTALS

As is known, fractal images seem to imitate nature rather well. Certain types of fractals employ a controlled random motion to produce images of landscapes, trees, etc. Mandelbrot calls these squig fractals.

Fractals have now worked their way into computer graphics art, and even into the movie theatre. The sharp landscape images now seen in some films were generated by taming Brownian (or random) motion. The techniques for generating these squig fractals are somewhat similar to those employed in generating the seed images. Again, an image must be drawn, and a data structure which anticipates the next level of iteration must be in place. However, the big difference is that in the production of squigs, a computerized "coin flip," provided by calling a random number routine, determines the final shape of the fractal.

Uncontrolled random motion appears too rough to model nature. This can be seen easily by randomly choosing (x, y) coordinates, and connecting the dots (see fig. 8). However, if some control is introduced into this process, such as scaling, transforming, or filtering, a pattern which imitates nature's own "controlled randomness" seems to emerge. This technique of creating images by controlled random motion has been employed successfully in the generation of landscape images, including mountain ranges and lakes (see fig. 9).

Unfortunately, the computer code to generate even the most basic of the squig images is long and tedious (listing 6, pp 34-40).

We can demonstrate the fundamental idea behind the creation of a squig fractal by discussing a planar squig curve. Given a simple closed polygonal curve, the squig procedure can be applied to produce a controlled Brownian path which circumvents nearly the same area as the initial curve.

Cover the initial curve with a grid of sufficiently small mesh size (say, d/4, where d = minimal distance between vertices). Let the length of one line segment in the grid be denoted by L. Align the mesh so that none of the polygon's vertices matches a vertex of the mesh. Thus, in each square that

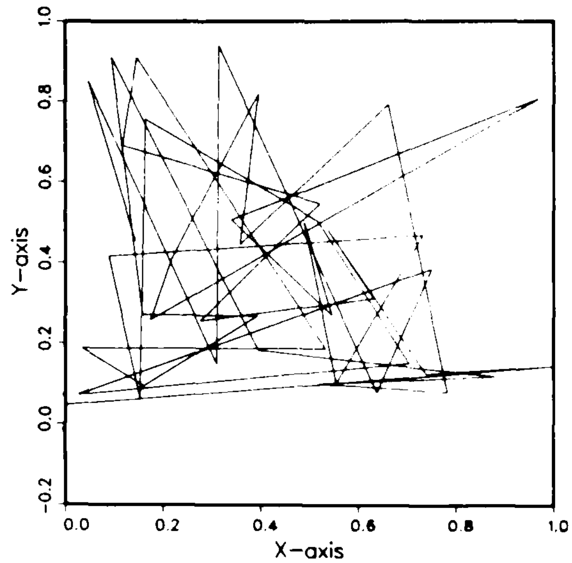


Figure 8. Brownian motion.

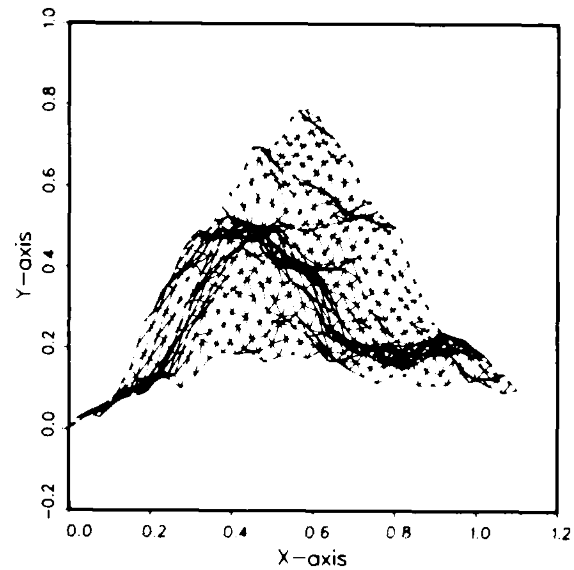


Figure 9. Skeletal fractal mountain.

the original path intersects, the polygonal path will come in one side and exit on another. Divide the box into four equal boxes having sides of length L/2. At this point, controlled random motion will produce the first generation of a squig curve.

Consider a single square. Assume that the path enters the box from the left. In the scale of the finer mesh (L/2), the path could enter the box from the top or bottom. In the squig scheme, this is determined by a weighted coin flip, e.g.,

$$\text{ran}(\ ) \le 0.25 = 3/4 \text{ weight} \quad .$$

If this is the first box processed on this level of iteration, then a fair coin flip (i.e., ran ( ) $\le$ 0.5) will determine entry. If not, then entry position is inherited from the previous box. Similarly, in the scale of the finer mesh, there are two choices of exit from each side. This is decided by a weighted coin flip. (This exit position then determines entry of the adjacent box.) Between entry and exit, there are 22 possible paths on the level of the finer mesh. There are eight exiting from the opposite side, and seven each from the adjacent sides. The choice of any of these paths comes from a couple of weighted coin flips, which determine whether or not the path turns or goes straight (see fig. 10).

Throughout the process, the coin flip is weighted in favor of the straighter path. The higher the weight, the straighter the path, and the lower the fractional dimension of the curve.

Once this process of choosing the path on the first level of iteration has been completed, the curve must be covered by boxes with sides of length L/4.

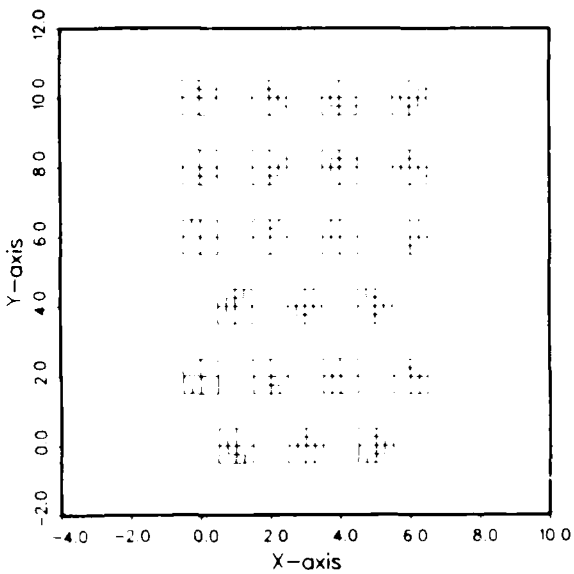


Figure 10. Possible two-dimensional squig paths.

In turn, each of these must be divided in equal fourths, and the next level of iteration in the squig procedure must be done. Iteration continues a predetermined number of times.

The squig procedure is by no means limited to curves. The idea of using chance to produce fractal behavior is a particularly powerful one. This is witnessed in The Fractal Geometry of Nature, for Mandelbrot devotes nearly the entire second half of the book to this subject. One particularly interesting use of the squig procedure is in the production of semirandom tree paths called "graftals" (see fig. 11). Estvanik's article on this subject (1986) is particularly useful.

Producing a squig is only the first step towards producing a realistic landscape. Smoothing, coloring, shading, shadowing, and numerous other more standard techniques from computer graphics must then be applied to produce the finished product, a realistic image. It is interesting to note, however, that because the squig (and other) fractal methods are so powerful, these techniques, once considered esoteric, are rapidly being accepted into the mainstream of computer graphics.

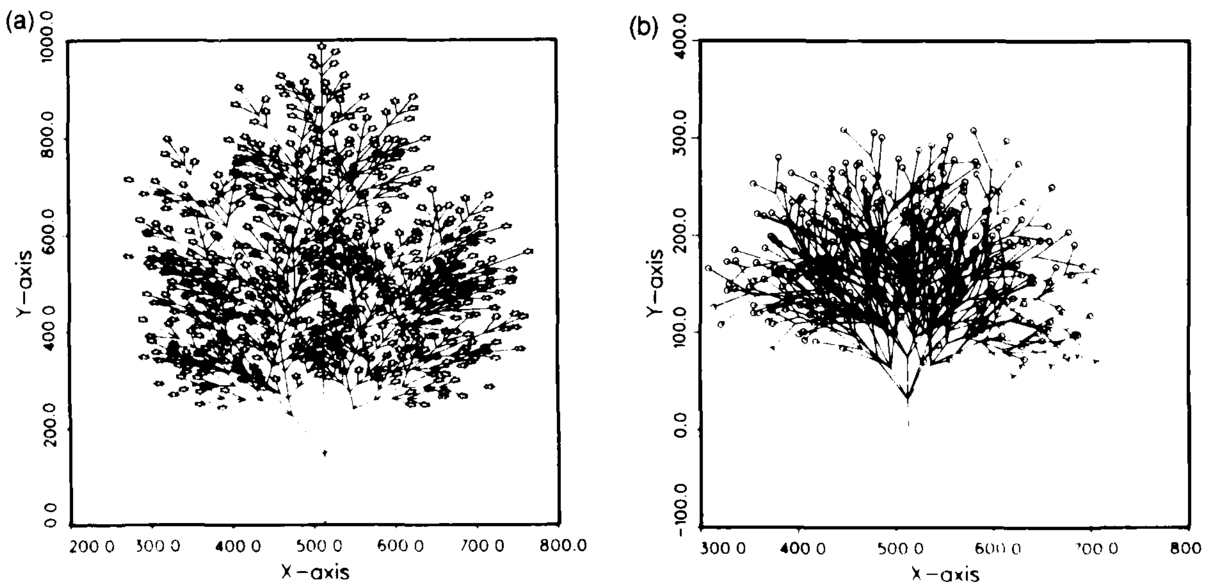


Figure 11. Graftal tree (a) and bush (b).

Listing 6. Code for plotting skeletal fractal mountains

```
c --------------------------------------------------------
c This program draws the skeleton of a fractal mountain
c  range by means of the addition of random weights to
c  the y values of a triangular lattice. The con-
c  struction of this lattice goes from 0 to Iter.
c Programmer : S. Casey
c --------------------------------------------------------
c

      PROGRAM MOUNTAIN

      REAL*8 X(0:6, 24768), Y(0:6, 24768)      !  Array of endpoints
      REAL*8 Mx(24768), My(24768)              !  Array of midpoints
      REAL*8 Randbound                         !  The degree of roughness
      REAL*8 Randshft   !  Function giving random number with random sign
      INTEGER*4 I, J, Iter, Level
      INTEGER*4 Depth(0:6), Numpts(0:6), Numlns(0:6), Numtrg(0:6)

      WRITE (6,*) ' Enter the # of iterates. Cannot do more than 6.'
      READ (5,*) Iter

      WRITE (6,*) ' Enter the degree of roughness. This is a number'
      WRITE (6,*) '     between 0 and 1.'
      READ (5,*) Randbound

      WRITE (6,*) ' Enter the initial triangle.'
      DO I = 1, 3
         READ (5,*) X(0, I), Y(0, I)
      ENDDO

      Depth(0) = 2
      Numpts(0) = 3
      Numlns(0) = 3
      Numtrg(0) = 1
      DO I = 1, Iter
         Depth(I) = Depth(I-1) + 2**(I-1)
         Numpts(I) = ( Depth(I) * (Depth(I) + 1) ) / 2
         Numlns(I) = 3 * ( ( (Depth(I) - 1) * Depth(I) ) / 2 )
         Numtrg(I) = 4**I
      ENDDO

      Level = 0
      DO I = 1, Iter
         Level = Level + 1

         CALL Midpoints( X, Y, Randbound, Level, Mx, My )

         CALL Reassign( X, Y, Level, Mx, My )

      ENDDO

      OPEN ( 10, FILE='MOUNTAIN.DAT', STATUS='NEW', ERR=998,
     *   IOSTAT=IOS, CARRIAGECONTROL='LIST' )

      CALL Draw( X, Y, Iter )

      CLOSE( 10 )
      STOP
998   WRITE (6,*) ' Error opening new file MOUNTAIN.DAT = ', IOS
      STOP
      END
```

Listing 6. Code for plotting skeletal fractal mountains (cont'd)

```
c----------------------------------------------------------------------
c -- This function returns a bounded random number
c     -Randbound/2**Level <= num <= Randbound/2**Level
c        with random sign.
c

      REAL*8 FUNCTION Randshft( Randbound, Level )

      REAL*8 Randbound                 !  Variable received from main
      REAL*8 Rand, Sgn, Scale          !  Local variables
      INTEGER*4 Iseed, Timeseed        !  For the random number generator
      INTEGER*4 Level                  !  Level of iteration in main routine
      LOGICAL First/.TRUE./   ! Flag which insures only one call of Timeseed

      IF ( First ) THEN
         Iseed = Timeseed()
         First = .FALSE.
      ENDIF

      IF ( RAN(Iseed) .LT. 0.5 ) THEN
         Sgn = -1.0
      ELSE
         Sgn = 1.0
      ENDIF

      Rand = RAN(Iseed) * Randbound
      Scale = 1.0 / (2.0**Level)

      Randshft = Sgn * Scale * Rand

      RETURN
      END

c----------------------------------------------------------------------
c -- This function is a sleazy trick which enables people with typing
c      disabilities, like me, to get a large odd integer for RAN().
c

      INTEGER*4 FUNCTION Timeseed()

      Timeseed = INT(SECNDS(0.0))
      IF ( MOD(Timeseed,2) .EQ. 0 ) Timeseed = Timeseed + 1
      RETURN
      END

c----------------------------------------------------------------------
c -- This subroutine calculates the midpoints of the lattice(level-1).
c

      SUBROUTINE Midpoints( X, Y, Randbound, Level, Mx, My )

      REAL*8 X(0:6, 24768), Y(0:6, 24768)  ! Endpoints, from main
      REAL*8 Randbound                     ! Variable received from main
      REAL*8 Mx(24768), My(24768)          ! Returned to main
      REAL*8 Randshft                      ! Random number with random sign
      REAL*8 Rand, X1, X2, Y1, Y2          ! Local variables
```

Listing 6. Code for plotting skeletal fractal mountains (cont'd)

```
      INTEGER*4 Level                          ! Level of iteration in main
      INTEGER*4 I, J, Depth, Int, Count        ! Local variables
      INTEGER*4 Leadpt(129), Jump(0:129)

      Depth = 2
      DO I = 1, (Level-1)
         Depth = Depth + 2**(I-1)
      ENDDO

      Leadpt(1) = 1
      Int = 0
      DO I = 2, (Depth+1)
         Int = Int + 1
         Leadpt(I) = Leadpt(I-1) + Int
      ENDDO

c -- Calculating midpoints of line segments "slanting left"

      Count = 0
      DO I = 1, (Depth-1)
         Jump(0) = 0
         DO J = 1, (Depth-I)
            Jump(J) = (J + I - 1) + Jump(J-1)
            X1 = X( (Level-1), ( (Leadpt(I+1)-1) + Jump(J-1) ) )
            Y1 = Y( (Level-1), ( (Leadpt(I+1)-1) + Jump(J-1) ) )
            X2 = X( (Level-1), ( (Leadpt(I+1)-1) + Jump(J) ) )
            Y2 = Y( (Level-1), ( (Leadpt(I+1)-1) + Jump(J) ) )
            Rand = Randshft( Randbound, Level )
            Count = Count + 1
            Mx(Count) = (X1 + X2) / 2.0
            My(Count) = ((Y1 + Y2) / 2.0) + Rand
         ENDDO
      ENDDO

c -- Calculating midpoints of line segments "slanting right"

      DO I = 1, (Depth-1)
         Jump(0) = 0
         DO J = 1, (Depth-I)
            Jump(J) = (J + I) + Jump(J-1)
            X1 =  X( (Level-1), (Leadpt(I) + Jump(J-1)) )
            Y1 =  Y( (Level-1), (Leadpt(I) + Jump(J-1)) )
            X2 =  X( (Level-1), (Leadpt(I) + Jump(J)) )
            Y2 =  Y( (Level-1), (Leadpt(I) + Jump(J)) )
            Rand = Randshft( Randbound, Level )
            Count = Count + 1
            Mx(Count) = (X1 + X2) / 2.0
            My(Count) = ((Y1 + Y2) / 2.0) + Rand
         ENDDO
      ENDDO
```

Listing 6. Code for plotting skeletal fractal mountains (cont'd)

```
c -- Calculating midpoints of "horizontal" line segments

      DO I = 2, Depth
         DO J = Leadpt(I), Leadpt(I+1)-2
            X1 = X( (Level-1), J )
            Y1 = Y( (Level-1), J )
            X2 = X( (Level-1), J+1 )
            Y2 = Y( (Level-1), J+1 )
            Rand = Randshft( Randbound, Level )
            Count = Count + 1
            Mx(Count) = (X1 + X2) / 2.0
            My(Count) = ((Y1 + Y2) / 2.0) + Rand
         ENDDO
      ENDDO

      RETURN
      END
c-----------------------------------------------------------------------
c -- This subroutine reassigns points of lattice(level-1) and new
c        midpoints to the lattice(level).
c

      SUBROUTINE Reassign( X, Y, Level, Mx, My )

      REAL*8 X(0:6, 24768), Y(0:6, 24768)   ! Array of endpoints
      REAL*8 Mx(24768), My(24768)           ! Array of midpoints

      INTEGER*4 Level                       ! Level of iteration in main
      INTEGER*4 I, J, Int, Jump, Count      ! Local variables
      INTEGER*4 Skip1, Skip2
      INTEGER*4 Depth(0:7), Leadpt(129)

      Depth(0) = 2
      DO I = 1, Level+1
         Depth(I) = Depth(I-1) + 2**(I-1)
      ENDDO

      Leadpt(1) = 1
      Int = 0
      DO I = 2, Depth(Level+1)
         Int = Int + 1
         Leadpt(I) = Leadpt(I-1) + Int
      ENDDO

c -- Reassigning old lattice points

      X( Level, 1 ) = X( (Level-1), 1 )
      Y( Level, 1 ) = Y( (Level-1), 1 )
      Jump = 0
      DO I = 2, Depth(Level-1)
         Jump = Jump + 1
         X( Level, Leadpt(I+Jump) ) = X( (Level-1), Leadpt(I) )
         Y( Level, Leadpt(I+Jump) ) = Y( (Level-1), Leadpt(I) )
         Skip1 = 0
         DO J = Leadpt(I)+1, Leadpt(I+1)-1
            Skip1 = Skip1 + 1
            X( Level, (Leadpt(I+Jump)+(2*Skip1)) ) = X( (Level-1), J )
            Y( Level, (Leadpt(I+Jump)+(2*Skip1)) ) = Y( (Level-1), J )
         ENDDO
      ENDDO
```

Listing 6. Code for plotting skeletal fractal mountains (cont'd)

```
c -- Reassigning new midpoints to lattice

c -- Reassigning midpoints of line segments slanting left

      Count = 0
      Skip1 = 0
      DO I = 1, (Depth(Level)-1)
         Skip1 = Skip1 + 1
         Jump = 0
         IF ( MOD(Skip1, 2) .NE. 0 ) THEN
            DO J = 1, (Depth(Level)-I)
               Jump = (J + I - 1) + Jump
               IF ( MOD(J, 2) .NE. 0 ) THEN
                  Count = Count + 1
                  X( Level, (Leadpt(I+1)-1+Jump) ) = Mx( Count )
                  Y( Level, (Leadpt(I+1)-1+Jump) ) = My( Count )
               ENDIF

            ENDDO
         ENDIF
      ENDDO

c -- Reassigning midpoints of line segemnts slanting right

      Skip1 = 0
      DO I = 1, (Depth(Level)-1)
         Skip1 = Skip1 + 1
         Jump = 0
         IF ( MOD(Skip1, 2) .NE. 0 ) THEN
            DO J = 1, (Depth(Level)-I)
               Jump = (J + I) + Jump
               IF ( MOD(J, 2) .NE. 0 ) THEN
                  Count = Count + 1
                  X( Level, (Leadpt(I)+Jump) ) = Mx( Count )
                  Y( Level, (Leadpt(I)+Jump) ) = My( Count )
               ENDIF
            ENDDO
         ENDIF
      ENDDO

c -- Reassigning midpoints of horizontal line segments

      Skip1 = 1
      Skip2 = 0
      DO I = 2, Depth(Level)
         Skip1 = Skip1 + 1
         IF ( MOD(Skip1, 2) .NE. 0 ) THEN
            DO J = Leadpt(I)+1, Leadpt(I+1)-1
               Skip2 = Skip2 + 1
               IF ( MOD(Skip2, 2) .NE. 0 ) THEN
                  Count = Count + 1
                  X( Level, J ) = Mx( Count )
                  Y( Level, J ) = My( Count )
               ENDIF
            ENDDO
         ENDIF
      ENDDO

      RETURN
      END
```

```
c------------------------------------------------------------------
c -- This subroutine draws the lattice(Iter).
c

      SUBROUTINE Draw( X, Y, Iter )

      REAL*8 X(0:6, 24768), Y(0:6, 24768)   ! Array of endpoints
      INTEGER*4 Iter
      INTEGER*4 I, J, Depth, Int, Jump
      INTEGER*4 Leadpt(129)
      CHARACTER*1 Lntyp        ! Draws either points(P) or vectors(V)

 100  FORMAT(1X,F10.3,1X,F10.3,1X,A)

      Depth = 2
      DO I = 1, Iter
         Depth = Depth + 2**(I-1)
      ENDDO

      Leadpt(1) = 1
      Int = 0
      DO I = 2, (Depth+1)
         Int = Int + 1
         Leadpt(I) = Leadpt(I-1) + Int
      ENDDO

c -- Drawing line segments "slanting left"

      DO I = 1, (Depth-1)
         Lntyp = 'P'
         WRITE(10,100) X( Iter, (Leadpt(I+1)-1) ),
     *                 Y( Iter, (Leadpt(I+1)-1) ), Lntyp
         Jump = 0
         DO J = 1, (Depth-I)
            Jump = (J + I - 1) + Jump
            Lntyp = 'V'
            WRITE(10,100) X( Iter, ( (Leadpt(I+1)-1) + Jump ) ),
     *                    Y( Iter, ( (Leadpt(I+1)-1) + Jump ) ),
     *                     Lntyp
         ENDDO
      ENDDO

c -- Drawing line segments "slanting right"

      DO I = 1, (Depth-1)
         Lntyp = 'P'
         WRITE(10,100) X( Iter, Leadpt(I) ),
     *                 Y( Iter, Leadpt(I) ), Lntyp
         Jump = 0
         DO J = 1, (Depth-I)
            Jump = (J + I) + Jump
            Lntyp = 'V'
            WRITE(10,100) X( Iter, (Leadpt(I) + Jump) ),
     *                    Y( Iter, (Leadpt(I) + Jump) ), Lntyp
         ENDDO
      ENDDO

c -- Drawing "horizontal" line segments

      DO I = 2, Depth
         Lntyp = 'P'
```

Listing 6. Code for plotting skeletal fractal mountains (cont'd)

```
      WRITE(10,100) X( Iter, Leadpt(I) ),
     *                 Y( Iter, Leadpt(I) ), Lntyp
      DO J = (Leadpt(I)+1), (Leadpt(I+1)-1)
         Lntyp = 'V'
         WRITE(10,100) X( Iter, J ),
     *                    Y( Iter, J ), Lntyp
      ENDDO
   ENDDO

   RETURN
   END
```

5. REMARKS ON APPLICATIONS

Because fractals apparently mimic nature so well, they have been applied to the study of numerous areas. Chemists, biologists, physicists, statiticians, etc, have been using fractals lately to model behavior in their particular fields. Fractals could even be applied to digital signal processing, as shown by the following argument.

Fractals are quasi-self-repeating images. Therefore, by definition, the image seen on one level is nearly mimicked by an enlargement on the next. Moreover, the only limitations to this magnification are the built-in limitations of the image-producing machines (the fractal itself will allow any magnification).

Therefore, aligned in the proper digital fashion, a fractal could be used as a communications code. The digitized fractal image would act like a carrier or an envelope along which information would travel. It should make a good code for the following two reasons:

Uniqueness--A fractal image has its own unique imprint. Small perturbations in the fractal can produce large variation. (This can be demonstrated by varying λ in the iteration of $z^2 + \lambda$.) Thus, properly chosen, a fractal can produce a unique digital sequence.

Stability (and instability)--Fractals should be extremely stable with respect to noise because of their quasi-self-repeating nature. To decide whether or not a bit is a good piece of information, the bit could be enlarged at a lower level. There, the criterion of the information's correctness can be predetermined by how far it is from the fractal's basic quasi-pattern. Further enlargements only improve upon the fractal's estimate.

ACKNOWLEDGEMENTS

I would like to thank Robert Miller for his work on graphics and especially his work on the Raster Technologies equipment.

Calculations were done on a Digital VAX 11/750. Color images were produced on a Raster Technologies Model ONE/80, and the black and white images were printed on a QMS Laser Grafix 1200 laser printer. Assistance in data storage was provided by James Griffin. Steven Choy helped with the laser printer. The color photographs were taken from the display screen by Lawrence Shank, who used Kodachrome 50 film at f4, 1/8 sec.

BIBLIOGRAPHY

Fractals

Dewdney, A. K. (1985, August). Computer Recreations, Scientific American, 253 No. 2, 16-24.

Estvanik, S. (1986, July). From Fractals to Graftals, Computer Language, 3, No. 7, 45-58.

Mandelbrot, B. B. (1983). The Fractal Geometry of Nature, W. H. Freeman and Company, New York.

Mathematics

Ahlfors, L. (1979). Complex Analysis, McGraw Hill, New York.

Benedetto, J. J. (1976). Real Variable and Integration, B. G. Teuber, Stuttgart.

Blanchard, P. (1984). Complex Analytic Dynamics on the Riemann Sphere, Bull. Am. Math. Soc., 11, No. 1, 85-141.

Guckenheimer, J., and P. Holmes (1983). Non-Linear Oscillations, Dynamical Systems, and Bifurcation of Vector Fields, Springer Verlag, New York.

Hurewicz, W., and H. Wallman (1948). Dimension Theory, Princeton University Press, Princeton.

Lehto, O., and K. I. Virtanen (1973). Quasiconformal Mappings in the Plane, Springer Verlag, Berlin.

Munkres, J. (1975). Topology, Prentice-Hall, New York.

DISTRIBUTION

ADMINISTRATOR
DEFENSE TECHNICAL INFORMATION CENTER
ATTN DTIC-DDA (12 COPIES)
CAMERON STATION, BUILDING 5
ALEXANDRIA, VA 22304-6145

US ARMY ELECTRONICS TECHNOLOGY
& DEVICES LABORATORY
ATTN DELET-DD
FT MONMOUTH, NJ 07703

DEPT OF THE AIR FORCE, H1
6585TH TEST GROUP (AFSC)
RADAR TARGET SCATTER FACILITY
ATTN LT COL RONALD L. KERCHER, CHIEF
HOLLMAN AFB, NM 88330

ENGINEERING SOCIETIES LIBRARY
ATTN ACQUISITIONS DEPARTMENT
345 EAST 47TH STREET
NEW YORK, NY 10017

INSTITUTE FOR TELECOMMUNICATIONS
SCIENCES
NATIONAL TELECOMMUNICATIONS
& INFO ADMIN
ATTN LIBRARY
BOULDER, CO 80303

DIRECTOR
DEFENSE COMMUNICATIONS AGENCY
ATTN TECH LIBRARY
ATTN COMMAND & CONTROL CENTER
WASHINGTON, DC 20305

DIRECTOR
DEFENSE COMMUNICATIONS ENGINEERING
CENTER
ATTN TECHNICAL LIBRARY
1860 WIEHLE AVE
RESTON, VA 22090

DIRECTOR
NASA
GODDARD SPACE FLIGHT CENTER
ATTN 250, TECH INFO DIV
GREENBELT, MD 20771

DIRECTOR
NASA
ATTN TECHNICAL LIBRARY
JOHN F. KENNEDY SPACE
CENTER, FL 32899

DIRECTOR
NASA
LANGLEY RESEARCH CENTER
ATTN TECHNICAL LIBRARY
HAMPTON, VA 23665

DIRECTOR
NASA
LEWIS RESEARCH CENTER
ATTN TECHNICAL LIBRARY
CLEVELAND, OH 44135

US ARMY LABORATORY COMMAND
ATTN TECHNICAL DIRECTOR, AMSLC-CT

INSTALLATION SUPPORT ACTIVITY
ATTN LIBRARY, SLCIS-IM-TL (3 COPIES)
ATTN LIBRARY, SLCIS-IM-TL (WOODBRIDGE)
ATTN LEGAL OFFICE, SLCIS-CC

USAISC
ATTN RECORD COPY, ASNC-ADL-TS
ATTN TECHNICAL REPORTS BRANCH,
ASNC-ADL-TR

HARRY DIAMOND LABORATORIES
ATTN D/DIVISION DIRECTORS
ATTN CHIEF, SLCHD-NW-E
ATTN CHIEF, SLCDH-NW-EC
ATTN CHIEF, SLCDH-NW-ED
ATTN CHIEF, SLCHD-NW-EE
ATTN CHIEF, SLCHD-NW-R
ATTN CHIEF, SLCHD-NW-RA
ATTN CHIEF, SLCHD-NW-RC
ATTN CHIEF, SLCHD-NW-RE
ATTN CHIEF, SLCHD-NW-RH
ATTN CHIEF, SLCHD-NW-RI
ATTN CHIEF, SLCHD-NW-P
ATTN R. GOODMAN, SLCHD-DE-OS
ATTN B. ZABLUDOWSKI, SLCHD-IT-EB
ATTN R. CHASE, SLCHD-NW-EC
ATTN C. KENYON, SLCHD-NW-EC
ATTN T. WYATT, SLCHD-NW-EC
ATTN H. BOESCH, SLCHD-NW-RC
ATTN F. MCLEAN, SLCHD-NW-RC
ATTN S. HAYES, SLCHD-NW-RE
ATTN S. CHOY, SLCHD-IT-R
ATTN N. BERG, SLCHD-RT-R
ATTN P. ALEXANDER, SLCHD-RT-AD
ATTN C. ARSEM, SLCHD-RT-AD
ATTN J. COSTANZA, SLCHD-RT-AD
ATTN J. DAMMANN, SLCHD-RT-AD
ATTN D. HULL, SLCHD-RT-AD
ATTN J. SELTZER, SLCHD-RT-AD
ATTN W. SHVODIAN, SLCHD-RT-AD
ATTN B. WEBER, SLCHD-RT-AC
ATTN D. WONG, SLCHD-RT AD
ATTN A. FILIPOV, SLCHD-RT-RB
ATTN C. GARVIN, SLCHD-RT-RB
ATTN D. GLENN, SLCHD-RT-RB
ATTN J. GRIFFIN, SLCHD-RT-RB
ATTN R. JOHNSON, SLCHD-RT-RB
ATTN N. KARAYIANIS, SLCHD-RT-RB
ATTN D. MCGUIRE, SLCHD-RT-RB

HARRY DIAMOND LABORATORIES (cont'd)
ATTN R. MILLER, SLCHD-RT-RB
ATTN B. SADLER, SLCHD-RT-RB
ATTN R. ULRICH, SLCHD-RT-RB
ATTN P. EMMERMAN, SLCHD-RT-RD (2 COPIES,
ATTN S. CASEY, SLCHD-RT-RB (100 COPIES)

END

JAN.

1988

DTIC